Database - Bug #18

issues with ne/equal and true/false conversion in WHERE clauses

09/26/2011 12:00 PM - Stanislav Lomany

Status:	WIP	**Start date:**	10/23/2012
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD		

Description

Related issues:

Related to Base Language - Bug #2288: incorrect conversion and results for un...	**Closed**

History

#1 - 09/19/2011 06:11 AM - Stanislav Lomany

@45197 - issue description:

```
The issue is that l <> true is incorrectly converted to _not(l) and l <> false - to (l).booleanValue().
Also, cases 8-1, 8-2, 8-4 and 8-5 in the testcase need to be checked.
```

#2 - 09/19/2011 06:13 AM - Redmine Admin

@45198 - Status changed from WIP to Hold

#3 - 09/26/2011 01:16 AM - Stanislav Lomany

@45175 - issue description:

```
<> true, <> false and = false are misconverted:

Testcase:

def temp-table tt
     field f as logical.

create tt. tt.f = true.
create tt. tt.f = false.
create tt. tt.f = ?.

def var l as logical.
def var l2 as logical init true.
def var l3 as logical init false.
def var l4 as logical init ?.

l = true.
if l = true then message "test1-1".
if l = false then message "test1-2".
if l = ? then message "test1-3".
if l = l2 then message "test1-4".
if l = l3 then message "test1-5".
if l = l4 then message "test1-6".
```

```
if l <> true then message "test2-1".
if l <> false then message "test2-2".
if l <> ? then message "test2-3".
if l <> l2 then message "test2-4".
if l <> l3 then message "test2-5".
if l <> l4 then message "test2-6".

l = false.
if l = true then message "test3-1".
if l = false then message "test3-2".
if l = ? then message "test3-3".
if l = l2 then message "test3-4".
if l = l3 then message "test3-5".
if l = l4 then message "test3-6".

if l <> true then message "test4-1".
if l <> false then message "test4-2".
if l <> ? then message "test4-3".
if l <> l2 then message "test4-4".
if l <> l3 then message "test4-5".
if l <> l4 then message "test4-6".

l = ?.
if l = true then message "test5-1".
if l = false then message "test5-2".
if l = ? then message "test5-3".
if l = l2 then message "test5-4".
if l = l3 then message "test5-5".
if l = l4 then message "test5-6".

if l <> true then message "test6-1".
if l <> false then message "test6-2".
if l <> ? then message "test6-3".
if l <> l2 then message "test6-4".
if l <> l3 then message "test6-5".
if l <> l4 then message "test6-6".

for each tt where f = true: message "test7-1". end.
for each tt where f = false: message "test7-2". end.
for each tt where f = ?: message "test7-3". end.
for each tt where f = l2: message "test7-4". end.
for each tt where f = l3: message "test7-5". end.
for each tt where f = l4: message "test7-6". end.

for each tt where f <> true: message "test8-1". end.
for each tt where f <> false: message "test8-2". end.
for each tt where f <> ?: message "test8-3". end.
for each tt where f <> l2: message "test8-4". end.
for each tt where f <> l3: message "test8-5". end.
for each tt where f <> l4: message "test8-6". end.
```

#4 - 04/12/2012 11:23 AM - Stanislav Lomany

Imported from JPRM on 2012-04-12 11:23:10.563:

```
TASKID      = 6119
PROJECTID   = 124
STATUS      = 3 (Hold)
DESCRIPTION = ne/equal  true/false conversion
OWNER       = SVL
ASSIGNEE    = SVL
BILLABLE    = false
PRIORITY    = 5 (Normal)
PHASE       = 52 (Problem Resolution)
COMPONENT   = 0
ESTEFFORT   = 0.000000
ESTSTART    = 2011-09-26
ESTSTOP     =
ACTSTOP     =
CASENUM     =
VENDORID    =
COMMENT     = ''
LASTWIP     = 2011-09-19
```

#5 - 10/23/2012 03:56 PM - Greg Shah

- Assignee changed from Stanislav Lomany to Vadim Nebogatov

#6 - 10/23/2012 03:57 PM - Greg Shah

- Start date set to 10/23/2012

#7 - 10/25/2012 05:35 AM - Vadim Nebogatov

Added
OUTPUT TO VALUE.
to output test results to file.

Tested on the customer's 4GL environment. Test results:

test1-1
test1-4
test2-2
test2-3
test2-5
test2-6
test3-2
test3-5
test4-1
test4-3
test4-4
test4-6

test5-3
test5-6
test6-1
test6-2
test6-4
test6-5
test7-1
test7-2
test7-3
test7-4
test7-5
test7-6
test8-1
test8-1
test8-2
test8-2
test8-3
test8-3
test8-4
test8-4
test8-5
test8-5
test8-6
test8-6

#8 - 10/29/2012 04:54 PM - Vadim Nebogatov

Place to be corrected is found. It is the rule
type prog.equals or type prog.not_eq) and (ancestor(prog.expression, -1) or ancestor(prog.kw_where, -1))
in rules/fixups/normalize_expressions.rules.

#9 - 10/31/2012 06:57 PM - Vadim Nebogatov

Reason is found: expressions

(l <> true) and (NOT l)
(l <> false) and (l)

are not equivalent in Postgres 4GL in contrast to equivalent expressions

(l = true) and (l)
(l = false) and (not l)

All rules in p2j/rules/fixups/normalize_expressions.rules related with replacement of logical values with prog.not_eq should be removed including ones containing in WHERE statement like
(in testing):

```
         <rule>
            (type == prog.equals and
```

```
            analog.getImmediateChild(prog.bool_true, null) != null
         ) and !ancestor(prog.kw_where, -1)
            <action>
               ref = analog.getImmediateChild(prog.bool_true, null)
            </action>
         <action>execLib("moveOppositeNode", analog, ref)</action>
         <action>modified = true</action>
      </rule>

      <rule>
         !modified and
         (type == prog.equals and
           analog.getImmediateChild(prog.bool_false, null) != null)
            <action>
               ref = analog.getImmediateChild(prog.bool_false, null)
            </action>
         <action>ref.remove()</action>
         <action>analog.type = prog.kw_not</action>
         <action>analog.text = "not"</action>
      </rule>

   </rule>

   <rule>
      type == prog.equals and
      parent.type == prog.expression                and
      parent.parent.type == prog.kw_where
      <action>lChild = this.getChildAt(0)</action>
      <action>rChild = this.getChildAt(1)</action>
      <rule>
         (lChild.type == prog.unknown_val and
          ((rChild.type == prog.func_recid and
           #(long) rChild.getAnnotation("oldtype") == prog.kw_recid) or
           (rChild.type == prog.func_rowid and
           #(long) rChild.getAnnotation("oldtype") == prog.kw_rowid))) or
         (rChild.type == prog.unknown_val and
          ((lChild.type == prog.func_recid and
           #(long) lChild.getAnnotation("oldtype") == prog.kw_recid) or
           (lChild.type == prog.func_rowid and
           #(long) lChild.getAnnotation("oldtype") == prog.kw_rowid)))

            <action>
               execLib("replaceNode", analog, prog.bool_false)
            </action>

      </rule>
   </rule>
```

#10 - 11/01/2012 05:12 AM - Greg Shah

- Status changed from New to WIP

Questions/requests/comments:

0. I think you mean Progress 4GL, not Postgres 4GL. <g>

1. Please modify the testcase such that test7-* and test8-* output results that can be interpreted. Since we don't know which of the temp-table records is actually matching the condition, the output for these cases is currently indeterminate. I suggest adding a column to the temp table that is a simple row number as in:

```
def temp-table tt
     field num as int
     field f as logical.

create tt. tt.num = 1. tt.f = true.
create tt. tt.num = 2. tt.f = false.
create tt. tt.num = 3. tt.f = ?.
```

Then change the test7 and test8 message statements to output the row number in addition to the current text.

2. It seems to me that part of the problem here is how the NOT operator works. Please enhance the testcase to add cases to show the full range of usage of the NOT operator on both sides of the = or <> operators. This will require 3 additional sets of tests for every current set of tests. For example, this code:

```
if l = true then message "test1-1".
if l = false then message "test1-2".
if l = ? then message "test1-3".
if l = l2 then message "test1-4".
if l = l3 then message "test1-5".
if l = l4 then message "test1-6".
```

would probably need to look like this:

```
if l = true then message "test1-1".
if l = false then message "test1-2".
if l = ? then message "test1-3".
if l = l2 then message "test1-4".
if l = l3 then message "test1-5".
if l = l4 then message "test1-6".

if (not l) = true then message "test1-7".
if (not l) = false then message "test1-8".
if (not l) = ? then message "test1-9".
if (not l) = l2 then message "test1-10".
if (not l) = l3 then message "test1-11".
if (not l) = l4 then message "test1-12".

if l = (not true) then message "test1-13".
if l = (not false) then message "test1-14".
if l = (not ?) then message "test1-15".
if l = (not l2) then message "test1-16".
if l = (not l3) then message "test1-17".
if l = (not l4) then message "test1-18".

if (not l) = (not true) then message "test1-19".
if (not l) = (not false) then message "test1-20".
if (not l) = (not ?) then message "test1-21".
if (not l) = (not l2) then message "test1-22".
if (not l) = (not l3) then message "test1-23".
if (not l) = (not l4) then message "test1-24".
```

3. Recapture the 4GL output of the updated test case. Post it here.

4. Capture the P2J output of the updated test case (without any changes to normalize_expressions.rules). Post it here.

5. When posting a potential fix for a problem, it is important to show both the original code and the proposed change. Your entry above was your proposed change, but it was not obvious by just reading the entry. I had to go to the original file and read the whole thing to determine that you were posting the proposed changes. In the future, show both so the reader can immediately get the idea of what the change means.

6. Capture the P2J output of the updated test case WITH your proposed changes to normalize_expressions.rules. Post it here.

#11 - 11/01/2012 05:14 AM - Greg Shah

I should have mentioned this too:

7. Make sure you post your updated version of the testcase code (the 4GL) here.

8. Cross reference the differences in 4GL and P2J output and post the list of input expressions that are mis-converted (when the proposed changes are not applied).

9. I presume that the changes completely eliminate all issues. If not, please note what is still different between 4GL and P2J after your changes are applied.

#12 - 11/01/2012 04:13 PM - Vadim Nebogatov

(not ?) - syntax is incorrect

#13 - 11/01/2012 04:14 PM - Vadim Nebogatov

Updated test case:

```
def temp-table tt
field num as int
field f as logical.

create tt. tt.num = 1. tt.f = true.
create tt. tt.num = 2. tt.f = false.
create tt. tt.num = 3. tt.f = ?.

def var l as logical.
def var l2 as logical init true.
def var l3 as logical init false.
def var l4 as logical init ?.

l = true.

if l = true then message "test1-1".
if l = false then message "test1-2".
if l = ? then message "test1-3".
if l = l2 then message "test1-4".
if l = l3 then message "test1-5".
if l = l4 then message "test1-6".

if (not l) = true then message "test1-7".
if (not l) = false then message "test1-8".
if (not l) = ? then message "test1-9".
if (not l) = l2 then message "test1-10".
if (not l) = l3 then message "test1-11".
if (not l) = l4 then message "test1-12".

if l = (not true) then message "test1-13".
if l = (not false) then message "test1-14".
if l = (not l2) then message "test1-15".
```

```
if l = (not l3) then message "test1-16".
if l = (not l4) then message "test1-17".

if (not l) = (not true) then message "test1-18".
if (not l) = (not false) then message "test1-19".
if (not l) = (not l2) then message "test1-20".
if (not l) = (not l3) then message "test1-21".
if (not l) = (not l4) then message "test1-22".

if l <> true then message "test2-1".
if l <> false then message "test2-2".
if l <> ? then message "test2-3".
if l <> l2 then message "test2-4".
if l <> l3 then message "test2-5".
if l <> l4 then message "test2-6".

if (not l) <> true then message "test2-7".
if (not l) <> false then message "test2-8".
if (not l) <> ? then message "test2-9".
if (not l) <> l2 then message "test2-10".
if (not l) <> l3 then message "test2-11".
if (not l) <> l4 then message "test2-12".

if l <> (not true) then message "test2-13".
if l <> (not false) then message "test2-14".
if l <> (not l2) then message "test2-15".
if l <> (not l3) then message "test2-16".
if l <> (not l4) then message "test2-17".

if (not l) <> (not true) then message "test2-18".
if (not l) <> (not false) then message "test2-19".
if (not l) <> (not l2) then message "test2-20".
if (not l) <> (not l3) then message "test2-21".
if (not l) <> (not l4) then message "test2-22".

l = false.

if l = true then message "test3-1".
if l = false then message "test3-2".
if l = ? then message "test3-3".
if l = l2 then message "test3-4".
if l = l3 then message "test3-5".
if l = l4 then message "test3-6".

if (not l) = true then message "test3-7".
if (not l) = false then message "test3-8".
if (not l) = ? then message "test3-9".
if (not l) = l2 then message "test3-10".
if (not l) = l3 then message "test3-11".
if (not l) = l4 then message "test3-12".

if l = (not true) then message "test3-13".
if l = (not false) then message "test3-14".
if l = (not l2) then message "test3-15".
if l = (not l3) then message "test3-16".
if l = (not l4) then message "test3-17".

if (not l) = (not true) then message "test3-18".
if (not l) = (not false) then message "test3-19".
if (not l) = (not l2) then message "test3-20".
if (not l) = (not l3) then message "test3-21".
if (not l) = (not l4) then message "test3-22".

if l <> true then message "test4-1".
if l <> false then message "test4-2".
if l <> ? then message "test4-3".
if l <> l2 then message "test4-4".
if l <> l3 then message "test4-5".
if l <> l4 then message "test4-6".

if (not l) <> true then message "test4-7".
if (not l) <> false then message "test4-8".
if (not l) <> ? then message "test4-9".
if (not l) <> l2 then message "test4-10".
if (not l) <> l3 then message "test4-11".
```

```
if (not l) <> l4 then message "test4-12".

if l <> (not true) then message "test4-13".
if l <> (not false) then message "test4-14".
if l <> (not l2) then message "test4-15".
if l <> (not l3) then message "test4-16".
if l <> (not l4) then message "test4-17".

if (not l) <> (not true) then message "test4-18".
if (not l) <> (not false) then message "test4-19".
if (not l) <> (not l2) then message "test4-20".
if (not l) <> (not l3) then message "test4-21".
if (not l) <> (not l4) then message "test4-22".

l = ?.

if l = true then message "test5-1".
if l = false then message "test5-2".
if l = ? then message "test5-3".
if l = l2 then message "test5-4".
if l = l3 then message "test5-5".
if l = l4 then message "test5-6".

if (not l) = true then message "test5-7".
if (not l) = false then message "test5-8".
if (not l) = ? then message "test5-9".
if (not l) = l2 then message "test5-10".
if (not l) = l3 then message "test5-11".
if (not l) = l4 then message "test5-12".

if l = (not true) then message "test5-13".
if l = (not false) then message "test5-14".
if l = (not l2) then message "test5-15".
if l = (not l3) then message "test5-16".
if l = (not l4) then message "test5-17".

if (not l) = (not true) then message "test5-18".
if (not l) = (not false) then message "test5-19".
if (not l) = (not l2) then message "test5-20".
if (not l) = (not l3) then message "test5-21".
if (not l) = (not l4) then message "test5-22".

if l <> true then message "test6-1".
if l <> false then message "test6-2".
if l <> ? then message "test6-3".
if l <> l2 then message "test6-4".
if l <> l3 then message "test6-5".
if l <> l4 then message "test6-6".

if (not l) <> true then message "test6-7".
if (not l) <> false then message "test6-8".
if (not l) <> ? then message "test6-9".
if (not l) <> l2 then message "test6-10".
if (not l) <> l3 then message "test6-11".
if (not l) <> l4 then message "test6-12".

if l <> (not true) then message "test6-13".
if l <> (not false) then message "test6-14".
if l <> (not l2) then message "test6-15".
if l <> (not l3) then message "test6-16".
if l <> (not l4) then message "test6-17".

if (not l) <> (not true) then message "test6-18".
if (not l) <> (not false) then message "test6-19".
if (not l) <> (not l2) then message "test6-20".
if (not l) <> (not l3) then message "test6-21".
if (not l) <> (not l4) then message "test6-22".

for each tt where f = true: message "test7-1 row=" num. end.
for each tt where f = false: message "test7-2 row=" num. end.
for each tt where f = ?: message "test7-3 row=" num. end.
for each tt where f = l2: message "test7-4 row=" num. end.
for each tt where f = l3: message "test7-5 row=" num. end.
for each tt where f = l4: message "test7-6 row=" num. end.
```

```
for each tt where f <> true: message "test8-1 row=" num. end.
for each tt where f <> false: message "test8-2 row=" num. end.
for each tt where f <> ?: message "test8-3 row=" num. end.
for each tt where f <> l2: message "test8-4 row=" num. end.
for each tt where f <> l3: message "test8-5 row=" num. end.
for each tt where f <> l4: message "test8-6 row=" num. end.
```

#14 - 11/01/2012 04:15 PM - Vadim Nebogatov

4GL results for updated test case:

```
test1-1
test1-4
test1-8
test1-11
test1-14
test1-16
test1-18
test1-20
test2-2
test2-3
test2-5
test2-6
test2-7
test2-9
test2-10
test2-12
test2-13
test2-15
test2-17
test2-19
test2-21
test2-22
test3-2
test3-5
test3-7
test3-10
test3-13
test3-15
test3-19
test3-21
test4-1
test4-3
test4-4
test4-6
test4-8
test4-9
test4-11
test4-12
test4-14
test4-16
test4-17
test4-18
test4-20
test4-22
test5-3
test5-6
test5-9
test5-12
test5-17
test5-22
test6-1
test6-2
test6-4
test6-5
test6-7
test6-8
test6-10
test6-11
test6-13
```

test6-14
test6-15
test6-16
test6-18
test6-19
test6-20
test6-21
test7-1 row= 1
test7-2 row= 2
test7-3 row= 3
test7-4 row= 1
test7-5 row= 2
test7-6 row= 3
test8-1 row= 2
test8-1 row= 3
test8-2 row= 1
test8-2 row= 3
test8-3 row= 1
test8-3 row= 2
test8-4 row= 2
test8-4 row= 3
test8-5 row= 1
test8-5 row= 3
test8-6 row= 1
test8-6 row= 2

#15 - 11/01/2012 07:20 PM - Vadim Nebogatov

P2J results for updated test case:

test1-1
test1-4
test1-8
test1-11
test1-14
test1-16
test1-18
test1-20
test2-2
test2-3
test2-5
test2-6
test2-7
test2-9
test2-10
test2-12
test2-13
test2-15
test2-17
test2-19
test2-21

test2-22
test3-2
test3-5
test3-7
test3-10
test3-13
test3-15
test3-19
test3-21
test4-1
test4-3
test4-4
test4-6
test4-8
test4-9
test4-11
test4-12
test4-14
test4-16
test4-17
test4-18
test4-20
test4-22
test5-3
test5-6
test5-9
test5-12
test5-17
test5-22
test6-4
test6-5
test6-10
test6-11
test6-15
test6-16
test6-20
test6-21
test7-1 row= 1
test7-2 row= 2
test7-3 row= 3
test7-4 row= 1
test7-5 row= 2
test7-6 row= 3
test8-1 row= 2
test8-2 row= 1
test8-3 row= 1
test8-3 row= 2
test8-4 row= 2
test8-5 row= 1
test8-6 row= 1
test8-6 row= 2

#16 - 11/02/2012 12:43 PM - Vadim Nebogatov

- File Screenshot.png added

Comparison results (see attached screenshot)

#17 - 11/05/2012 10:23 AM - Greg Shah

Do the proposed changes fully eliminate all differences?

#18 - 11/06/2012 07:25 AM - Vadim Nebogatov

All Test6 problems are fixed, Test8 problems are not fixed yet.

#19 - 11/06/2012 04:42 PM - Vadim Nebogatov

Problems are with Test8 are related with the following "where" clauses used in AdaptiveQuery constructor:

tt.ff !=false, tt.ff !=true, tt.f != ?

For database fields comparison A!=B should be replaced with

(A is null and B is not null) or (A is not null and B is null) or (A!=B)

#20 - 11/06/2012 05:19 PM - Greg Shah

Eric: should this change go into the HQLPreprocessor instead of the conversion? The resulting code seems very messy.

#21 - 11/06/2012 05:42 PM - Eric Faulhaber

Greg Shah wrote:

> Eric: should this change go into the HQLPreprocessor instead of the conversion? The resulting code seems very messy.

I'm not sure. What would the converted code look like before it was expanded by the HQLPreprocessor? And more importantly, do we want to oversimplify and hide what's really going on?

#22 - 11/06/2012 06:03 PM - Greg Shah

I don't know. I guess that is for you and Vadim to discuss further.

#23 - 11/06/2012 06:15 PM - Eric Faulhaber

I would expect that this is not a very common case, so my inclination is to fix it in the conversion and leave the more expressive HQL intact. If we find it is more prevalent than I expect, we can revisit the idea of implementing in the HQLPreprocessor, but I am reluctant to hide this away.

#24 - 11/07/2012 11:42 AM - Vadim Nebogatov

Small correction:

A!=B

should be generated as

(A is null and B is not null) or (A is not null and (B is null or A!=B))

This expression is correct for all possible values, so after substitution we should generate

1) for Test8-1, Test8-4 (comparison with true)

(tt.f is null and true is not null) or (tt.f is not null and (true is null or tt.f!=true))

2) for Test8-2, Test8-5 (comparison with false)

(tt.f is null and false is not null) or (tt.f is not null and (false is null or tt.f!=false))

3) for Test8-3, Test8-6 (comparison with ?)

(tt.f is null and null is not null) or (tt.f is not null and (null is null or tt.f!=null))

Of course, expressions are not optimal, but correct. After I made these changes in already converted *.java code, test results became fully identical with 4GL

#25 - 11/07/2012 05:15 PM - Vadim Nebogatov

- File normalize_expressions.rules added

Changed normalize_expressions.rules is attached (Test6 problems).

#26 - 11/07/2012 08:43 PM - Eric Faulhaber

It would be better to test for and avoid emitting things like:

- true is null
- true is not null
- false is null
- false is not null
- null is null
- null is not null

These seem to be special cases that should drop out. Is there any reason they could not?

#27 - 11/08/2012 09:37 AM - Greg Shah

Feedback on the normalize_expressions.rules changes:

1. You have removed the first line of the file, which should be there:

```
<?xml version="1.0"?>
```

2. The copyright date needs update.

3. Your change is missing a history entry.

4. I would like you to document that we are deliberately NOT normalizing the NE FALSE and NE TRUE cases. Put comments in the 2 locations where you changed the code. It should explain why those optimizations cannot be done.

Otherwise, the code changes themselves look fine.

#28 - 11/08/2012 01:25 PM - Vadim Nebogatov

- File normalize_expressions.rules added

Attached variant seems ready to commit (when CVS is unlocked)

#29 - 11/08/2012 01:43 PM - Greg Shah

I agree that this file looks good. So it passes code review.

BUT:

1. There are more issues to resolve before this task is done. Usually, on small tasks we don't check in partial fixes.
2. We need to regression test this change (and the rest of your pending changes, whatever they are) using a full conversion project that is in production today (TIMCO's Majic application). To do this we convert with and without your change and compare the results. If there are no unexpected changes, then we are OK.
3. If this change does change the Majic code or if we made changes that otherwise affect the runtime (e.g. to the P2J runtime code), then we would run full functional regression testing on Majic. This is automated testing that takes about 4 hours.

Only after all of the above, will it be approved for check in. But that doesn't mean it will be checked in immediately. We are very careful with the order of our changes. So we will queue it up to be applied to CVS and to the TIMCO "staging" environment. Once applied to both places, you would send out an update zip file with the changes. This makes it easier for people to maintain testing environments that cannot simply check out the changes from CVS (e.g. TIMCO's staging directory on their servers).

Also, normally you would post the complete set of changes as an update zip file and attach that to the issue in Redmine. That is something new in our process, previously we sent a "proposed update" email with the zip at code review time. Now we just attach the zip for code review.

Much of these details about our process are documented in the developer guide and the coding standards. Please do review those carefully.

By the way, we are making updates to those documents right now so the process will be changing slightly soon. But learning the current process is a good thing.

#30 - 11/08/2012 02:02 PM - Vadim Nebogatov

Concerning special cases

```
true is null
    true is not null
    false is null
    false is not null
    null is null
    null is not null.
```

I also think these hardcode cases should be replaced with corresponding values.

#31 - 11/08/2012 02:07 PM - Greg Shah

Please go ahead with optimizing these cases (e.g. true is null can be replaced with false).

#32 - 11/09/2012 06:07 PM - Vadim Nebogatov

One more, pure database field test:

def temp-table tt
field f1 as logical
field f2 as logical.

create tt. tt.f1 = true. tt.f2 = true.
create tt. tt.f1 = true. tt.f2 = false.
create tt. tt.f1 = true. tt.f2 = ?.
create tt. tt.f1 = false. tt.f2 = true.
create tt. tt.f1 = false. tt.f2 = false.
create tt. tt.f1 = false. tt.f2 = ?.
create tt. tt.f1 = ?. tt.f2 = true.
create tt. tt.f1 = ?. tt.f2 = false.
create tt. tt.f1 = ?. tt.f2 = ?.

for each tt where f1 <> f2: message "NE: f1=" f1 "f2=" f2. end.

4GL result:

NE: f1= yes f2= no
NE: f1= yes f2= ?
NE: f1= no f2= yes
NE: f1= no f2= ?
NE: f1= ? f2= yes
NE: f1= ? f2= no

P2J result:

NE: f1= yes f2= no
NE: f1= no f2= yes

After replacement
"tt.f1 != tt.f2"
to
"(tt.f1 is null and tt.f2 is not null) or (tt.f1 is not null and (tt.f2 is null or not (tt.f1=tt.f2)))"
in generated java file, results are the same as in 4GL.
This expression could be simplified in special cases like discussed above, but in common case seems could not be simplified. Expression "not (tt.f1 = tt.f2)" gives the same results like "tt.f1 != tt.f2"

#33 - 11/15/2012 06:04 PM - Vadim Nebogatov

Based on idea above, new rule and function "replaceInequalityNode" are added. Function seems is not optimal, I think also that variables could be reused, but test result is now the same as in 4GL.

```
&lt;rule&gt;
         (type == prog.not_eq) and
         ancestor(prog.kw_where, -1)
         &lt;action&gt;lChild = this.getChildAt(0)&lt;/action&gt;
         &lt;action&gt;rChild = this.getChildAt(1)&lt;/action&gt;
          &lt;action&gt;execLib("replaceInequalityNode", analog, lChild, rChild)&lt;/action&gt;
          &lt;action&gt;modified = false&lt;/action&gt;
      &lt;/rule&gt;



&lt;function name="replaceInequalityNode"&gt;
           &lt;parameter name="dstref" type="com.goldencode.ast.Aast" /&gt;
           &lt;parameter name="lRef"   type="com.goldencode.ast.Aast" /&gt;
           &lt;parameter name="rRef"   type="com.goldencode.ast.Aast" /&gt;
           &lt;variable  name="orRef"   type="com.goldencode.ast.Aast" /&gt;
           &lt;variable  name="andRef1"   type="com.goldencode.ast.Aast" /&gt;
           &lt;variable  name="andRef2"   type="com.goldencode.ast.Aast" /&gt;
           &lt;variable  name="lRefCopy"   type="com.goldencode.ast.Aast" /&gt;
           &lt;variable  name="rRefCopy"   type="com.goldencode.ast.Aast" /&gt;
           &lt;variable  name="equalsRef1"   type="com.goldencode.ast.Aast" /&gt;
           &lt;variable  name="inequalsRef1"   type="com.goldencode.ast.Aast" /&gt;
           &lt;variable  name="equalsRef2"   type="com.goldencode.ast.Aast" /&gt;
           &lt;variable  name="inequalsRef2"   type="com.goldencode.ast.Aast" /&gt;
           &lt;variable  name="orRef2"   type="com.goldencode.ast.Aast" /&gt;
           &lt;variable  name="equalsRef3"   type="com.goldencode.ast.Aast" /&gt;
           &lt;variable  name="notRef"   type="com.goldencode.ast.Aast" /&gt;
           &lt;variable  name="equalsRef4"   type="com.goldencode.ast.Aast" /&gt;
           &lt;variable  name="lparens1"   type="com.goldencode.ast.Aast" /&gt;



&lt;rule&gt;lRef==null
                  &lt;rule&gt;rRef==null
                        &lt;action&gt;orRef = createProgressAst(prog.kw_false, null)&lt;/action&gt;
                        &lt;action&gt;execLib("moveNode", dstref, orRef)&lt;/action&gt;
                  &lt;/rule&gt;
                  &lt;rule&gt;rRef!=null
                         &lt;action&gt;orRef = createProgressAst(prog.kw_true, null)&lt;/action&gt;
                         &lt;action&gt;execLib("moveNode", dstref, orRef)&lt;/action&gt;
                  &lt;/rule&gt;
         &lt;/rule&gt;
         &lt;rule&gt;lRef!=null
              &lt;rule&gt;rRef==null
                      &lt;action&gt;orRef = createProgressAst(prog.kw_true, null)&lt;/action&gt;
                      &lt;action&gt;execLib("moveNode", dstref, orRef)&lt;/action&gt;
             &lt;/rule&gt;
             &lt;rule&gt;rRef!=null
                  &lt;!-- main OR --&gt;
                  &lt;action&gt;orRef = createProgressAst(prog.kw_or, null)&lt;/action&gt;
                  &lt;action&gt;execLib("moveNode", dstref, orRef)&lt;/action&gt;



&lt;!-- first AND --&gt;
                  &lt;action&gt;andRef1 = createProgressAst(prog.kw_and, orRef, 0)&lt;/action&gt;
                  &lt;action&gt;equalsRef1 = createProgressAst(prog.equals, andRef1, 0)&lt;/action&gt;
                  &lt;action&gt;lRefCopy = lRef.duplicateFresh()&lt;/action&gt;
                  &lt;action&gt;equalsRef1.graft(lRefCopy)&lt;/action&gt;
                  &lt;action&gt;createProgressAst(prog.unknown_val, equalsRef1, 1)&lt;/action&gt;
                  &lt;action&gt;inequalsRef1 = createProgressAst(prog.not_eq, andRef1, 1)&lt;/action&gt;
                  &lt;action&gt;rRefCopy = rRef.duplicateFresh()&lt;/action&gt;
                  &lt;action&gt;inequalsRef1.graft(rRefCopy)&lt;/action&gt;
                  &lt;action&gt;createProgressAst(prog.unknown_val, inequalsRef1, 1)&lt;/action&gt;
                  &lt;!-- second AND --&gt;
                  &lt;action&gt;andRef2 = createProgressAst(prog.kw_and, orRef, 1)&lt;/action&gt;
                  &lt;action&gt;inequalsRef2 = createProgressAst(prog.not_eq, andRef2, 0)&lt;/action&gt;
                  &lt;action&gt;lRefCopy = lRef.duplicateFresh()&lt;/action&gt;
                  &lt;action&gt;inequalsRef2.graft(lRefCopy)&lt;/action&gt;
                  &lt;action&gt;createProgressAst(prog.unknown_val, inequalsRef2, 1)&lt;/action&gt;
```

```
                &lt;action&gt;lparens1 = createProgressAst(prog.lparens, "(", andRef2, 1)&lt;/action&gt;
                &lt;!-- OR --&gt;
                &lt;action&gt;orRef2 = createProgressAst(prog.kw_or, lparens1, 0)&lt;/action&gt;
                &lt;action&gt;equalsRef3 = createProgressAst(prog.equals, orRef2, 0)&lt;/action&gt;
                &lt;action&gt;rRefCopy = rRef.duplicateFresh()&lt;/action&gt;
                &lt;action&gt;equalsRef3.graft(rRefCopy)&lt;/action&gt;
                &lt;action&gt;createProgressAst(prog.unknown_val, equalsRef3, 1)&lt;/action&gt;
                &lt;action&gt;notRef = createProgressAst(prog.kw_not, orRef2, 1)&lt;/action&gt;
                &lt;action&gt;equalsRef4 = createProgressAst(prog.equals, notRef, 0)&lt;/action&gt;
                &lt;action&gt;lRefCopy = lRef.duplicateFresh()&lt;/action&gt;
                &lt;action&gt;equalsRef4.graft(lRefCopy)&lt;/action&gt;
                &lt;action&gt;rRefCopy = rRef.duplicateFresh()&lt;/action&gt;
                &lt;action&gt;equalsRef4.graft(rRefCopy)&lt;/action&gt;
            &lt;/rule&gt;
        &lt;/rule&gt;
    &lt;/function&gt;
```

#34 - 11/16/2012 05:40 PM - Vadim Nebogatov

Problem found. In common case is that not enough to change

expr1 != expr2

to

expr1 is null and expr2 is not null) or (expr1 is not null and (expr2 is null or not (expr1=expr2))

but we need also to correct args passed to AdaptiveQuery: some of args should be repeated.

#35 - 11/17/2012 04:26 PM - Vadim Nebogatov

Variant with changes in QUERY_SUBST is investigated. It is also possible solution but not very easy because QUERY_SUBST contains parameters from both expressions expr1 and expr2.

But it seems I have found better solution related with IS DISTINCT FROM predicate: http://blog.jooq.org/2012/09/21/the-is-distinct-from-predicate. Not all databases supports this predicate, in particular, Oracle does not support. Last link contains list of databases

SUPPORTED:

- Firebird
- H2
- HSQLDB
- Postgres

NOT SUPPORTED:

- CUBRID
- DB2
- Derby
- Ingres
- Oracle

- SQL Server
- SQLite
- Sybase ASE
- Sybase SQL Anywhere

I have tested with H2: it works. PostgreSQL supports this predicate starting from version 8.0 (http://www.postgresql.org/docs/8.0/static/functions-comparison.html)

I am not sure if rules are best place to change "!=" to " IS DISTINCT FROM ". Maybe we should patch WHERE clause in PreselectQuery.Component constructor? Of course we should not patch quoted strings containing "!="

#36 - 11/18/2012 05:42 PM - Eric Faulhaber

We can only use constructs that can be expressed in HQL and which are supported across all major databases (e.g., Oracle, PostgreSQL, SQL Server, DB2, MySQL, plus H2).

#37 - 11/19/2012 09:41 AM - Eric Faulhaber

Vadim Nebogatov wrote:

> Problem found. In common case is that not enough to change
>
> expr1 != expr2
>
> to
>
> expr1 is null and expr2 is not null) or (expr1 is not null and (expr2 is null or not (expr1=expr2))
>
> but we need also to correct args passed to AdaptiveQuery: some of args should be repeated.

Please post an example of this (the original 4GL plus the required, converted AdaptiveQuery), so I can better understand this issue.

#38 - 11/19/2012 05:45 PM - Vadim Nebogatov

In common case 4GL inequality expression in WHERE clause

exp1!=exp2

should be replaced with

(exp1 is null and exp2 is not null) or (exp1 is not null and (exp2 is null or not (exp1=exp2)))

It is equivalent of IS DESTINCT FROM.

AST processing of such inequalities should be done in 2 steps

a) rule for converting exp1!=exp2 to long expression above
b) rules for simplification of this expressions skipping subexpressions containing hql=false elements

Situation with QUERY_SUBST looks simpler than I expected initially: we need only repeat QUERY_SUBST expressions list 3 (three) times in the same order .
Namely 3 because subexpressions order in expression is like in initial expression expr1!=expr2

expr1, expr2, expr1,expr2, expr1,expr2

Example 1.

```
def temp-table tt
field f1 as logical
field f2 as logical.

def var l1 as logical init true.
def var l2 as logical init false.

...
for each tt where (f1 = l1) <> (f2 = l2): message "NE". end.
```

Expression now is "(f1 = l1) != (f2 = l2)" and QUERY_SUBST consists on 2 expressions: l1 and l2.

It should be replaced with expression

((f1 = l1) is null and (f2 = l2) is not null) or ((f1 = l1) is not null and ((f2 = l2) is null or not ((f1 = l1)=(f2 = l2))))
with QUERY_SUBST consisting on 6 expressions: l1, l2, l1, l2, l1, l2

Example 2

```
def temp-table tt
field f1 as logical.

def var l1 as logical init true.

...
for each tt where (f1 = l1) <> ?: message "NE". end.
```

Expression now is "tt.f1 = ? is not null"QUERY_SUBST consists on one expression: l1.
Should be
((tt.f1 = ?) is null and ? is not null) or ((tt.f1 = ?) is not null and (? is null or not ((tt.f1 = ?)=?)))

with QUERY_SUBST consisting on 3 expressions: l1, l1, l1

Last expression could be simplified with replacing constant subexpressions like “? is not null” and “? is null”:
((tt.f1 = ?) is null) or ((tt.f1 = ?) is not null and (not ((tt.f1 = ?)=?)))In common case 4GL inequality expression in WHERE clause

exp1!=exp2

should be replaced with

(exp1 is null and exp2 is not null) or (exp1 is not null and (exp2 is null or not (exp1=exp2)))

It is equivalent of IS DESTINCT FROM.

AST processing of such inequalities should be done in 2 steps

a) rule for converting exp1!=exp2 to long expression above
b) rules for simplification of this expressions skipping subexpressions containing hql=false elements

Situation with QUERY_SUBST looks simpler that I expected initially: we need only repeat QUERY_SUBST expressions list 3 (three) times in the same order .
Namely 3 because subexpressions order in expression is like in initial expression expr1!=expr2

expr1, expr2, expr1,expr2, expr1,expr2

Example 1.

```
def temp-table tt
field f1 as logical
field f2 as logical.

def var l1 as logical init true.
def var l2 as logical init false.
```

...
for each tt where (f1 = l1) <> (f2 = l2): message "NE". end.

Expression now is "(f1 = l1) != (f2 = l2)" and QUERY_SUBST consists on 2 expressions: l1 and l2.

It should be replaced with expression

((f1 = l1) is null and (f2 = l2) is not null) or ((f1 = l1) is not null and ((f2 = l2) is null or not ((f1 = l1)=(f2 = l2))))
with QUERY_SUBST consisting on 6 expressions: l1, l2, l1, l2, l1, l2

Example 2

def temp-table tt
field f1 as logical.

def var l1 as logical init true.

...
for each tt where (f1 = l1) <> ?: message "NE". end.

Expression now is "tt.f1 = ? is not null"QUERY_SUBST consists on one expression: l1.
Should be
((tt.f1 = ?) is null and ? is not null) or ((tt.f1 = ?) is not null and (? is null or not ((tt.f1 = ?)=?)))

with QUERY_SUBST consisting on 3 expressions: l1, l1, l1

Last expression could be simplified with replacing constant subexpressions like “? is not null” and “? is null”:
((tt.f1 = ?) is null) or ((tt.f1 = ?) is not null and (not ((tt.f1 = ?)=?)))

It should be simplified with separate rule

#39 - 11/20/2012 07:52 AM - Greg Shah

This should not be exploded into the code. It should be hidden in the HQL preprocessor.

There is even a good reason to only pass the query substitutions once (and then reuse the evaluated results multiple times): if there are side-effects to the evaluation, then the P2J version would execute multiple times, causing different results than in the 4GL.

Eric?.

#40 - 11/22/2012 12:21 AM - Eric Faulhaber

Yes, I agree, let's put it in the HQLPreprocessor.

#41 - 11/23/2012 05:15 PM - Vadim Nebogatov

I have found out that not only inequality should be replaced but equality also work incorrectly in some cases.
Very first test in this issue passed with equalities because of simplification rules, but test below – not (there are no simplifications in this test).

```
def temp-table tt
field f1 as logical
field f2 as logical.

create tt. tt.f1 = true. tt.f2 = true.
create tt. tt.f1 = true. tt.f2 = false.
create tt. tt.f1 = true. tt.f2 = ?.
create tt. tt.f1 = false. tt.f2 = true.
create tt. tt.f1 = false. tt.f2 = false.
create tt. tt.f1 = false. tt.f2 = ?.
create tt. tt.f1 = ?. tt.f2 = true.
create tt. tt.f1 = ?. tt.f2 = false.
create tt. tt.f1 = ?. tt.f2 = ?.

for each tt where f1 = f2: message "NE: f1=" f1 "f2=" f2. end.
```

4GL results:

```
NE: f1= yes f2= yes
NE: f1= no f2= no
NE: f1= ? F2= ?
```

P2J results

```
NE: f1= yes f2= yes
NE: f1= no f2= no
```

It should be expected because not(A=B) works like (A!=B), with the same errors.
So, we need to replace not only inequalities but also equalities, like described in
http://stackoverflow.com/questions/10416789/how-to-rewrite-is-distinct-from-and-is-not-distinct-from

Replacement should be nested, but inequalities and equalities received as result of replacement should not be processed again.
Parameter list for substitution should be also changed after each replacement.

#42 - 11/23/2012 05:34 PM - Vadim Nebogatov

What about limitations for SQL statement length which some databases have? Resulting SQL statement could be very heavy.

#43 - 11/23/2012 05:53 PM - Greg Shah

I'll let Eric comment on all the other good information and questions you have posed.

The only thing I want to make sure about is this: the query substitution parameters list should not be extended in the converted code. Where the list must be extended (with duplicate values), please do all that duplication inside the runtime code (e.g. the HQLPreprocessor). That way, there is no explosion of extra parameters in the generated code AND there is no way to evaluate the expressions (which may have side-effects) more than the same number of times that the expressions would have been evaluated in the 4GL.

#44 - 11/23/2012 06:01 PM - Vadim Nebogatov

Yes, I understand this. Converted code will be not affected in this case, HQLAst and parameter list changes should be in runtime code

#45 - 11/24/2012 05:00 PM - Eric Faulhaber

Vadim Nebogatov wrote:

> What about limitations for SQL statement length which some databases have? Resulting SQL statement could be very heavy.

Can IS [NOT] DISTINCT FROM be passed through HQL with the latest Hibernate version?

If so, perhaps this in an option after all, since in the HQLPreprocessor we have more flexibility with regard to varying the HQL by dialect. This could be the preferred mechanism for dialects which support it. If it is not supported by a dialect, and that dialect also has trouble with the length of the preprocessed HQL/SQL, I'm not sure how we deal with it.

#46 - 11/25/2012 12:25 PM - Vadim Nebogatov

IS [NOT] DISTINCT FROM predicate is not supported with Hibernate and it seems they even do not plan to support it:

https://nhibernate.jira.com/browse/NH-2614
• Resolution: Won't Fix

This predicate is included to ANSI SQL-99, so possible some future database releases will support it nevertheless.
BTW, MySQL supports analogous operator <=> for EQUAL (http://dev.mysql.com/doc/refman/5.0/en/comparison-operators.html#operator_equal-to).

What do you think about adding methods

String getNullSafeEqualToOperator()
String getNullSafeInequalToOperator()

to P2JDialect interface?

Method will return operator or null if such null safe equal operator is not supported.
For databases like MSQL, supporting only null safe equal operator, HQLPreprocessor could use NOT for NOT_EQ

#47 - 11/25/2012 02:04 PM - Eric Faulhaber

Vadim Nebogatov wrote:

> IS [NOT] DISTINCT FROM predicate is not supported with Hibernate and it seems they even do not plan to support it:
>
> https://nhibernate.jira.com/browse/NH-2614
> • Resolution: Won't Fix

I noticed this Jira item is for NHibernate, not Hibernate. However, if you have found that Hibernate does not support this syntax currently, that is really what matters, and we need to find another way.

> This predicate is included to ANSI SQL-99, so possible some future database releases will support it nevertheless.
> BTW, MySQL supports analogous operator <=> for EQUAL (
> http://dev.mysql.com/doc/refman/5.0/en/comparison-operators.html#operator_equal-to).
>
> What do you think about adding methods
>
> String getNullSafeEqualToOperator()
> String getNullSafeInequalToOperator()
>
> to P2JDialect interface?
>
> Method will return operator or null if such null safe equal operator is not supported.
> For databases like MSQL, supporting only null safe equal operator, HQLPreprocessor could use NOT for NOT_EQ

I don't remember this being a supported operator in HQL. Have you tested that Hibernate will pass this syntax through to the SQL it generates from our HQL?

BTW, I don't want to worry too much about MySQL, it is not currently a target for us. The vendors we currently are committed to support are PostgreSQL, MS SQL Server, and H2. Oracle is likely to be next.

I notice your implementation uses OR rather than the CASE syntax suggested here: http://wiki.postgresql.org/wiki/Is_distinct_from. Is there a reason you chose one over the other?

In any case, we need an implementation that can work on every dialect we need to support. Let's lock that down first. Everything else is just a "nice-to-have" optimization which may or may not work better for certain dialects. Which brings us back to your concern about lengthy SQL. Is that concern more than theoretical at this point? Have you hit such limits in your testing? What *are* the limits for the dialects we currently need to support?

#48 - 11/25/2012 04:56 PM - Vadim Nebogatov

Yes, you are correct, I did not notice that issue concerns NHibernate, but all others I wrote about Hibernate.

CASE or OR. The purpose was to minimize duplication of parameters. Example from Postgres wiki will require 4 substitutions of each parameter
But probably we could use combination of CASE and OR: it will require even less substitutions

CASE
WHEN (expr1 IS NULL) THEN (expr2 IS NULL)
ELSE ((expr2 IS NULL) OR (expr1 != expr2))
END

Example from Postgres wiki seems not correct: it returns NULL! According
http://www.postgresql.org/docs/8.2/static/functions-comparison.html

“For non-null inputs, IS DISTINCT FROM is the same as the <> operator. However, when both inputs are null it will return false, and when just one input is null it will return true. Similarly, IS NOT DISTINCT FROM is identical to = for non-null inputs, but it returns true when both inputs are null, and false when only one input is null. ”

My concerns about SQL statement size: I received such errors earlier when inserted long varchar (SQL statement is too long - msg#00160), but possible it was with old PostgreSQL (or driver) versions. It seems it is not so actually now:

Oracle9i
64K
http://docs.oracle.com/cd/B10501_01/server.920/a96536/ch44.htm

SQL Server
65,536 * Network packet size (The default packet size is 4 kilobytes)
http://msdn.microsoft.com/en-us/library/ms143432%28v=sql.90%29.aspx

PostgreSQL
Prior to 7.0, postgres only supported sql statements that were 8K long (or actually blocksize long which could be from 8K - 32K, but is 8K by default). That restriction was removed from the server in 7.0, however early versions on the 7.0 driver still had this restriction. This bug was fixed in later 7.0 versions of the jdbc driver.
http://archives.postgresql.org/pgsql-jdbc/2001-10/msg00364.php

H2
There is no limit for the following entities, except the memory and storage capacity: maximum identifier length (table name, column name, and so on); maximum number of tables, columns, indexes, triggers, and other database objects; maximum statement length, number of parameters per statement, tables per statement, expressions in order by, group by, having, and so on; maximum rows per query; maximum columns per table, columns per index, indexes per table, lob columns per table, and so on; maximum row length, index row length, select row length; maximum length of a varchar column, decimal column, literal in a statement.
But:
Limit on the complexity of SQL statements. Statements of the following form will result in a stack overflow exception:
SELECT * FROM DUAL WHERE X = 1
OR X = 2 OR X = 2 OR X = 2 OR X = 2 OR X = 2
-- repeat previous line 500 times --
http://www.h2database.com/html/advanced.html?highlight=limitation&search=limitation#limits_limitations

// I don't remember this being a supported operator in HQL.
What operator did you mean? HQL does not support <=> operator.

#49 - 11/26/2012 12:05 AM - Eric Faulhaber

Vadim Nebogatov wrote:

> I don't remember this being a supported operator in HQL.
>
> What operator did you mean? HQL does not support <=> operator.

Yes, I meant <=> operator.

OK, based on your research, it sounds like we shouldn't have a problem with SQL length for the databases we currently support or plan to support soon.

However, I am very concerned about what this expansion of the equality/inequality expression will do to performance. Equality tests are everywhere in converted queries. Inequality tests less so, but there are some. We can avoid the extra null testing for database fields we know are non-nullable (unfortunately there are very few in our first project) and for parameters we know are or are not null at the time of HQL preprocessing (there is already code in HQLPreprocessor that determines this).

Nevertheless, I'm still very concerned about the performance impact of the remaining expansions, especially for reports which have tons of queries in tight loops. We never have found a production case where the current processing caused a problem (that we know of). If we suddenly introduce an adverse impact to the performance of our first customer's production system, they will have a hard time accepting it.

I'm not sure what to do about this. I guess the length of regression testing should give us some indication of the performance hit. We'll need a baseline measurement of the time it takes a regression test run before the changes are introduced, and a measurement after, to have a rough idea.

#50 - 11/26/2012 06:22 PM - Vadim Nebogatov

I agreed absolutely. But it seems there is no choice if you want to fix this issue on the HQL level.
My draft idea was to make such processing at the beginning, and mainWalk() should remove all newly appeared obvious comparisons like NULL IS NULL, NULL IS NOT NULL and so on.

I have found additionally: compare result could depend on current database settings.

MSSQL: ANSI_NULLS flag
http://msdn.microsoft.com/en-en/library/ms188048.aspx (will be removed in future versions, also see Remarks)

PostgreSQL: transform_null_equals flag

http://www.postgresql.org/docs/8.3/static/runtime-config-compatible.html#GUC-TRANSFORM-NULL-EQUALS

Both these flags affect only direct comparisons with NULL like expr=NULL, and does not affect comparisons expr1=expr2 in common case.

But I think in any case using native IS [NOT] DISTINCT FROM for H2 and PostgreSQL would be preferable for these databases (and clearer to customers who use these databases intensively, PostgreSQL especially). Also such approach will require less testing and no changes in substitution parameters.

#51 - 11/27/2012 04:57 PM - Vadim Nebogatov

One more option is providing a custom comparing functions (org.hibernate.dialect.function.SQLFunction) registered with the Dialect. Not sure how it will affect performance

#52 - 11/27/2012 06:03 PM - Vadim Nebogatov

How to proceed with this task?..

#53 - 11/27/2012 06:21 PM - Eric Faulhaber

Please explain your SQLFunction idea more fully. It is intriguing.

I think this is a relatively new feature in Hibernate (we are currently still on v3.0.5). Does this entail defining an HQL function which is then translated to an arbitrary SQL fragment (like a dialect-specific expansion to IS DISTINCT FROM, or one of the more verbose variants of this we've been discussing)? Or must the output be an SQL function (which would be opaque to the database optimizer)?

If it is the former (i.e., arbitrary SQL fragment), would you please provide an example of how you would expect to implement this for this particular problem?

#54 - 11/29/2012 04:20 PM - Vadim Nebogatov

SQLFunction approach (SQLFunctions are already in Hibernate v3.0.5).

I have added

```
registerFunction("p2jnoequals", new InEqualsFunction());
      registerFunction("p2jequals", new EqualsFunction());
```

to P2JH2Dialect constructor, and created the following static inner classes:

```
private static abstract class SafeCompareFunction
{
      public Type getReturnType(Type columnType, Mapping mapping) throws QueryException
   {
         return Hibernate.BOOLEAN;
      }


public boolean hasArguments()
{
         return true;
      }


public boolean hasParenthesesIfNoArguments()
{
         return true;
      }
   }


private static class InEqualsFunction extends SafeCompareFunction implements SQLFunction
{
```

```
      public String render(List args, SessionFactoryImplementor factory) throws QueryException
   {
         if (args.size() != 2) {
            throw new QueryException("Function requires two arguments");
         }
         Object firstArg = args.get(0);
         Object secondArg = args.get(1);



//      What I'm trying to do:
//        (CASE
//        WHEN (?1 IS NULL) THEN (?2 IS NOT NULL)
//        ELSE ((?2 IS NULL) OR (?1 != ?2))
//        END)

StringBuilder buf = new StringBuilder();
         buf.append("(CASE WHEN (").append(firstArg).append(" IS NULL) THEN (").
                 append(secondArg).append(" IS NOT NULL) ELSE ((").append(secondArg).
                 append(" IS NULL) OR (").append(firstArg).append(" != ").append(secondArg).append(")) END)");
         return buf.toString();
      }



}



private static class EqualsFunction extends SafeCompareFunction implements SQLFunction
{
      public String render(List args, SessionFactoryImplementor factory) throws QueryException
   {
         if (args.size() != 2) {
            throw new QueryException("Function requires two arguments");
         }
         Object firstArg = args.get(0);
         Object secondArg = args.get(1);



//      What I'm trying to do:
//        (CASE
//        WHEN (?1 IS NULL) THEN (?2 IS NULL)
//        ELSE ((?2 IS NOT NULL) AND (?1 = ?2))
//        END)

StringBuilder buf = new StringBuilder();
         buf.append("(CASE WHEN (").append(firstArg).append(" IS NULL) THEN (").
                 append(secondArg).append(" IS NULL) ELSE ((").append(secondArg).
                 append(" IS NOT NULL) AND (").append(firstArg).append(" = ").append(secondArg).append(")) END
)");
         return buf.toString();
      }
   }
```

Example 1.

```
def temp-table tt
field f1 as logical
field f2 as logical.

create tt. tt.f1 = true. tt.f2 = true.
create tt. tt.f1 = true. tt.f2 = false.
create tt. tt.f1 = true. tt.f2 = ?.
create tt. tt.f1 = false. tt.f2 = true.
create tt. tt.f1 = false. tt.f2 = false.
create tt. tt.f1 = false. tt.f2 = ?.
create tt. tt.f1 = ?. tt.f2 = true.
create tt. tt.f1 = ?. tt.f2 = false.
create tt. tt.f1 = ?. tt.f2 = ?.

for each tt where f1 = f2: message ": f1=" f1 "f2=" f2. end.
```

Result of conversion

"tt.f1 = tt.f2"

replaced manually with

"p2jequals(tt.f1,tt.f2)"

Results are as expected

: f1= yes f2= yes
: f1= no f2= no
: f1= ? f2= ?

Example 2.

for each tt where f1 <> f2: message ": f1=" f1 "f2=" f2. End.

Result of conversion

"tt.f1 != tt.f2"

replaced manually with

"p2jnoequals(tt.f1,tt.f2)"

Results are as expected

: f1= yes f2= no
: f1= yes f2= ?
: f1= no f2= yes
: f1= no f2= ?
: f1= ? f2= yes
: f1= ? f2= no

Example 3. More complicated example with nested condition.

for each tt where (((f1 <> true) AND (f2 <> false)) <> (f2 <> ?)) : message ": f1=" f1 "f2=" f2. End.

Result of conversion

"tt.f1 != true and tt.f2 != false != tt.f2 is not null"

replaced manually with

"p2jnoequals(p2jnoequals(tt.f1, true) and p2jnoequals(tt.f2, false), tt.f2 is not null)"

Results:

: f1= no f2= no
: f1= no f2= ?
: f1= ? f2= no
: f1= ? f2= ?

Results are as in 4GL:

: f1= yes f2= yes
: f1= yes f2= no
: f1= no f2= no
: f1= no f2= ?
: f1= ? f2= no
: f1= ? f2= ?

So, this test did not pass so far. Analyzing is in progress. Maybe I replaced wrongly or there are some more details.

#55 - 11/29/2012 04:24 PM - Vadim Nebogatov

SQLFunctionTemplate does not allow repeating parameters, but direct implementation of SQLFunction interface seems resolves such problem. So we don't need to make any changes in parameters

#56 - 12/03/2012 04:54 PM - Vadim Nebogatov

Concerning Example 3:

I tested with H2 database: for case f1=true, f2=true expression

(tt.f1 != true and tt.f2 != false) != (tt.f2 is not null)

in where clause is equivalent to TRUE.

But converted code returns FALSE possible because of some not quite correct simplifications.

So, problem with Example 3 is not relevant to SQLFunction approach itself.

What is your decision? How should we move forward?

#57 - 12/04/2012 12:13 AM - Eric Faulhaber

OK, it appears you have determined the SQLFunction approach is feasible. Let's go with it and see how it performs. Perhaps we can shorten the function names to equals and not_equals, or even eq and neq? Greg: any input on this?

Please avoid adding this complexity as appropriate in cases where we know that:

- a field could never be null (i.e., it is mandatory/non-nullable);
- a substitution parameter value is known to be null or not null (we know this by the time the where clause hits the HQLPreprocessor and in fact Stanislav's earlier work in this area is what began this current issue).

One more thing to consider: what implications does this new functionality have on the existing caching strategy within HQLPreprocessor?

#58 - 12/04/2012 12:23 AM - Eric Faulhaber

Eric Faulhaber wrote:

> ...or even eq and neq?

I just remembered eq and ne already are user defined function names, so we need to avoid these. I'm trying to recall under which circumstances we use these in converted where clauses, if at all. I know I tried to avoid them because they are opaque to the optimizer, but it may be that we found we had to use them in certain cases. They do deal with unknown/null properly, but they are unusable on a wide scale because of the performance impact.

#59 - 12/04/2012 08:22 AM - Greg Shah

equals and not_equals seem fine to me.

I didn't realize that we already had eq and ne. When are they used? I previously thought that we always directly mapped = and <> to SQL operators.

#60 - 12/04/2012 08:23 AM - Greg Shah

Did I miss this? Is there a performance penalty to using the SQLFunction approach?

Also, what is the performance penalty for explicitly including the expanded SQL logic (versus our old incorrect approach)?

#61 - 12/04/2012 12:14 PM - Eric Faulhaber

Greg Shah wrote:

> I didn't realize that we already had eq and ne. When are they used? I previously thought that we always directly mapped = and <> to SQL operators.

I'm not sure we use these anymore. At one point, I had tried mapping all equality/inequality matches to these to ensure complete Progress compatibility, but the performance was abysmal, since the optimizer could not use any indexes. So, we reverted to the SQL operators again. There may be some edge cases where these functions are still in use.

#62 - 12/04/2012 12:29 PM - Eric Faulhaber

Greg Shah wrote:

> Did I miss this?

Apparently, yes, you missed the post (# 49) where I voiced my concern about performance.

> Is there a performance penalty to using the SQLFunction approach?

Probably. Consider that we are taking nearly every equality and inequality match and expanding it into a significantly more complex expression (or at best, into an IS [NOT] DISTINCT FROM phrase, which will have an unknown -- but likely not negligent -- impact on performance). If the database's optimizer can no longer apply indexes to the query plans because of this added complexity, the performance impact could be devastating, and this approach will have to be backed out.

Even if the optimizer properly can apply indexes to the most important (i.e., performance sensitive) portions of the where clause, I expect there to be a (hopefully minor) performance penalty simply because the query plans will be more complex (extra conditions/tests to apply). This would be a more subtle impact, and I am hoping it is the way things will work out, but we can't know for sure until we test the new approach.

In any event, I can't see performance improving from this, or even staying the same. My hope is that the degradation will not be noticeable.

> Also, what is the performance penalty for explicitly including the expanded SQL logic (versus our old incorrect approach)?

Same as above, since this essentially is what the rendering of the SQLFunction is doing at runtime. Or did I misunderstand your question?

#63 - 12/04/2012 06:05 PM - Vadim Nebogatov

Investigation SQLFunctions processing in Hibernate&Antlr source code. SqlGenerator invokes SQLFunction.render() method during final generation of SQL statement. Namely in this method we can try to generate different renderings based on arguments (null/not null). Not clear so far how to simplify this rendering for non-nullable fields. Maybe we should use for this SessionFactoryImplementor argument passed as argument for render().

#64 - 12/05/2012 01:28 PM - Greg Shah

Eric: my questions were not related to the potential worry that you have about performance issues. I was fully aware of your worry. My question is really: have we done any work so far to determine the performance cost? If not, when and how will we do that?

#65 - 12/05/2012 02:00 PM - Eric Faulhaber

AFAIK, we have not done any work to determine the potential performance penalty.

We will have to look at some typical queries in a real app (i.e., Majic), determine what the old and new SQL is, and use psql (i.e., EXPLAIN [ANALYZE]) to see what query plans PostgreSQL comes up with for both cases. This is something that needs a "real" database, with large tables and real-world selectivity statistics, since PostgreSQL will come up with radically different plans for "toy" databases (i.e., everything is a table scan). This is something that can be done before the implementation is complete, since we just need to know the differences in the SQL. It requires that we look at the SQL P2J/Hibernate generates today for these queries, then modify them to include what the SQLFunction implementation would do. It is important that we use prepared statements in psql, since that is what the P2J runtime does.

As to when: the sooner the better, as the results may impact Vadim's approach (at least for the PostgreSQL dialect). Greg: do you think it makes sense to wait for devsrv01 to be available, or to get Vadim an account on lightning, or to have someone else do this?

I guess we can go about broader testing once Vadim's proposed implementation is ready by running the regression harness with and without this feature and comparing the time it takes to run. This is less scientific, but it should at least be an indicator of whether we've really blown it. We can also revisit some of the more problematic reports in Majic and see if their times have gotten worse.

#66 - 12/05/2012 02:39 PM - Greg Shah

As far as when to do it, I would say to do a quick and dirty test soon if possible. You want to know now if there is an obvious/major problem.

A lot also depends on how close Vadim is to finishing the implementation. If it is just a 1 - 2 days away, then using the regression testing for this should be fine.

#67 - 12/05/2012 04:20 PM - Vadim Nebogatov

I also think it is hard to predict now how will SQLFunction approch affect caching and performance. I hope not very much.
I will try to complete this in 2 days. I also think that there is a sense to compare 2 variants of SQLFunctions implementation:

1) based on

```
?1!=?2 =>
            (CASE
            WHEN (?1 IS NULL) THEN (?2 IS NOT NULL)
            ELSE ((?2 IS NULL) OR (?1 != ?2))
            END)

?1=!2 =>
            (CASE
            WHEN (?1 IS NULL) THEN (?2 IS NULL)
```

```
            ELSE ((?2 IS NOT NULL) AND (?1 = ?2))
            END)
```

2) based on

```
IS [NOT] DISTINCT FROM
```

I tried to find out also how to simplify first approach for constant expressions and for not-nullable fields. The problem is that SQLFunction parameters are passed as strings on rendering time, this parameters could also contaon expressions containing both nullable and not-nullable fields. So on this stage we can only simplity if one of arguments is hardcode string like “null”, “true”, “false”.

#68 - 12/07/2012 12:50 PM - Eric Faulhaber

Vadim Nebogatov wrote:

> I tried to find out also how to simplify first approach for constant expressions and for not-nullable fields. The problem is that SQLFunction parameters are passed as strings on rendering time, this parameters could also contaon expressions containing both nullable and not-nullable fields. So on this stage we can only simplity if one of arguments is hardcode string like “null”, “true”, “false”.

See HQLPreprocessor.resolveBuffer(HQLAst). We could do something similar, given a string {dmo_alias}.{property_name}. Once we have the RecordBuffer object associated with the dmo alias name, we can ask it for its database and DMO class. Given this information, we can determine whether a particular property (i.e., column) is non-nullable from the DatabaseManager. Currently, this is exposed through the static notNullProperties(...) method, but perhaps we would need to expose this information slightly differently for a more efficient lookup in this case.

Are the strings representing database fields passed into SQLFunction as parameters in the form {dmo_alias}.{property_name} or {table_name}.{column_name}?

#69 - 12/07/2012 05:26 PM - Vadim Nebogatov

The arguments passed to SQLFunction are strings; we can parse them separately to Ast tree like method parse() does:

```
         StringReader reader = new StringReader((String)firstArg);
         HQLLexer lexer = new HQLLexer(reader);
```

```
        HQLParser parser = new HQLParser(lexer);
        parser.expression();
        HQLAst root = (HQLAst) parser.getAST();
        root.fixups(null, null);
```

(with convert parser exceptions to QueryException)

And simplify render() if root node is ALIAS and corresponding column is non-nullable.
What do you think? I checked this and it works (I checked only parsing so far and analyzing root nodes).

#70 - 12/10/2012 11:37 AM - Eric Faulhaber

Vadim Nebogatov wrote:

> The arguments passed to SQLFunction are strings; we can parse them separately to Ast tree like method parse() does:
>
> [...]
>
> (with convert parser exceptions to QueryException)
>
> And simplify render() if root node is ALIAS and corresponding column is non-nullable.
> What do you think? I checked this and it works (I checked only parsing so far and analyzing root nodes).

I think this is the right approach generally, though I don't know all the different forms the arguments can take.

...and from email:

> As I wrote I parse each argument of sql function to separate Ast and try to simplify render() if root node is ALIAS and corresponding column is non-nullable. But in this time aliases already look like <temprecord0_> while BufferManager contains mapping based on initial aliases like tt.

This is definitely a problem. In the P2J persistence framework, we do not deal with these aliases that Hibernate already has adjusted. Unless you can quickly figure out a way to map this back to our original aliases/buffers, let's defer the business about optimizing the expressions for non-nullable fields for now. Just optimize the constants (i.e., the parameters which are null, true, or false) for now and we'll figure out where to go from there.

#71 - 12/11/2012 04:44 PM - Vadim Nebogatov

- File sqlFunctions.tar.gz added

I have attached SQL Functions I am going to test now. Sorry, there are no proper comments and formatting just because I think that will be some changes after you look at this.

EFunction/NFunction are common equal/inequal functions with common rendering
DEFunction/DNFunction are equal/inequal functions with rendering based on IS [NOT] DISTINCT FROM, could be used for databases supporting this predicate. I am not sure, possible we also need to simplify null, true, false, not-nullable columns like it is made for EFunction/NFunction

SQL functions could be added to corresponding Dialect with

```
registerFunction("noeq", new NFunction());
      registerFunction("equal", new EFunction());
      registerFunction("dnoeq", new DNFunction());
      registerFunction("dequal", new DEFunction());
```

#72 - 12/12/2012 05:11 PM - Vadim Nebogatov

- File test_results.odt added

Some tests are attached and passed as expected.
Now I am working with tests containing nested sql functions - not all of them pass so far.

#73 - 12/14/2012 07:25 AM - Vadim Nebogatov

I understood the reason of problem with nested SQL Functions invocation (NPE on parsing SQL Function argument time): at this time we parse not HQL but already SQL statement.
One more problem is that we make this parsing each time on scroll(...) that affect caching and performance very much.

I am working at the following solution that should fix both these problems. Two more arguments will be added to SQL Function besides two main string arguments of the type from enum:

private static enum ArgType {NULL, TRUE, FALSE, NOT_NULLABLE, NULLABLE};

(this enum could be extended for future).
Types of each main argument will be calculated on HQLPreprocessor time, when we replace equals/inequals to SQL Functions. So, in scroll() time, when SQL Function parser() in invoked, only pure substitution will happen with simplifications based on ArgTypes of both string arguments.
I hope to complete this in 1-2 days

#74 - 12/15/2012 05:21 PM - Vadim Nebogatov

Approach described above is implemented, it works and on testing now.
But I have found one more problem.
If where clause is

(f1 <> true) <> (f1 <> f2)

P2j conversion result is

(A): "tt.f1 != true != tt.f1 != tt.f2"

But it is not the same as expected

(B): "(tt.f1 != true) != (tt.f1 != tt.f2)"

In case of string (B), SQL functions based results are the same as in 4GL:

NE: f1= yes f2= no
NE: f1= yes f2= ?
NE: f1= no f2= no
NE: f1= ? F2= ?

But in case of string (A) results are wrong

NE: f1= yes f2= no
NE: f1= yes f2= ?
NE: f1= no f2= no
NE: f1= no f2= ?
NE: f1= ? f2= no
NE: f1= ? F2= ?

#75 - 12/17/2012 06:02 PM - Vadim Nebogatov

- File test_results.odt added

Variant above does not require aliases mapping because argument type is calculated in HQLPreprocessor.
I made one more change in /home/vmn/p2j/rules/annotations/where_clause_normalize.rules:
added
...
copy.firstChild.type != prog.equals and
copy.firstChild.type != prog.not_eq

to the rule with comments
<!-- remove superfluous parentheses; it is safe to rely on the test
...

It seems it fixes problem with parentheses which I wrote above.
Tests are retested and attached, please check latest test especially. I plan to add some more complicated tests with non-nullable fields.

#76 - 12/19/2012 04:34 PM - Vadim Nebogatov

Reading and investigation of Progress 4GL documentation as regards not-nullable fields. I did not find any field options responsible for nullable/not-nullable property. Is such field option supported in 4GL? How make field as not-nullable?

#77 - 12/19/2012 04:35 PM - Vadim Nebogatov

Progress uses the keyword MANDATORY to denote a non-nullable field.

Thanks,
Eric

#78 - 12/19/2012 05:59 PM - Vadim Nebogatov

I read about this option, but I found no examples of it's using. Maybe it depends on 4GL version?
Syntax like

...
field f1 as logical mandatory
...

supported neither 4GL nor P2J

#79 - 12/20/2012 08:45 AM - Greg Shah

The keyword is specified in the schema (.df file) as an option on a field:

```
ADD FIELD "my-field" OF "some-table" AS character
  DESCRIPTION "Some random field."
  FORMAT "x(30)"
  INITIAL ""
  LABEL "My Field"
  POSITION 2
  SQL-WIDTH 30
  ORDER 20
  MANDATORY
```

It can only be specified for fields in a permanent database (non-temp-table). In the 4GL at runtime the state of that flag can be queried (but not set) via the MANDATORY attribute on a buffer-field handle.

#80 - 01/10/2013 06:08 PM - Vadim Nebogatov

Created sync testing environments for using persistent tables with mandatory/not-mandatory fields on 4GL and P2J: created testing database "vtest"

in 4GL with one table "btest" with 4 logical fields:

bn1, bn2 - not mandatory ("n" - nullable)
bnn1, bnn2 - mandatory ("nn" - not nullable)

Table is filled with all possible combinations

DEFINE VARIABLE a_bn1 AS LOGICAL EXTENT 3 INITIAL [TRUE, FALSE , ?]. /* optional */

DEFINE VARIABLE a_bn2 AS LOGICAL EXTENT 3 INITIAL [TRUE, FALSE , ?]. /* optional */

DEFINE VARIABLE a_bnn1 AS LOGICAL EXTENT 2 INITIAL [TRUE, FALSE]. /* mandatory */

DEFINE VARIABLE a_bnn2 AS LOGICAL EXTENT 2 INITIAL [TRUE, FALSE]. /* mandatory */

DEFINE VARIABLE i_bn1 AS INTEGER.

DEFINE VARIABLE i_bn2 AS INTEGER.

DEFINE VARIABLE i_bnn1 AS INTEGER.

DEFINE VARIABLE i_bnn2 AS INTEGER.

DO i_bn1 = 1 TO 3:

```
DO i_bn2 = 1 TO 3:


DO i_bnn1 = 1 TO 2:


DO i_bnn2 = 1 TO 2:


create btest. btest.bn1 = a_bn1[i_bn1]. btest.bn2 = a_bn2[i_bn2]. btest.bnn1 = a_bnn1[i_bnn1]. btest.bnn2 =a_b
nn2[i_bnn2].


END.


END.


END.
```

END.

Using FileZilla, copied vtest.df and btest.d to p2j, schema and table data are imported, first test is executed with imported database – everything works.

I have started equal/unequal tests with SQL Functions using persistent tables with mandatory fields.

#81 - 01/14/2013 04:47 PM - Vadim Nebogatov

- File test_results.odt added

I have attached tests passed for mandatory and not mandatory fields and their combinations

#82 - 01/14/2013 05:11 PM - Vadim Nebogatov

- File deleted (test_results.odt)

#83 - 01/14/2013 05:11 PM - Vadim Nebogatov

- File test_results.odt added

#84 - 01/14/2013 06:16 PM - Vadim Nebogatov

- File changed.tar.gz added

Changed sources are attached just for backup

#85 - 04/21/2014 01:34 PM - Eric Faulhaber

- Project changed from Bugs to Database

#86 - 04/21/2014 01:35 PM - Eric Faulhaber

- Subject changed from ne/equal true/false conversion to issues with ne/equal and true/false conversion in WHERE clauses

- Assignee deleted (Vadim Nebogatov)

- Target version set to Milestone 11

#87 - 04/21/2014 01:37 PM - Eric Faulhaber

The non-WHERE clause related conversion errors are being managed in #2288. The scope of this task is now just the WHERE clause related issues.

#88 - 06/02/2014 12:15 PM - Eric Faulhaber

- Target version deleted (Milestone 11)

This task is not known to affect any customer code; removing from M11 accordingly. If testing indicates otherwise, we will add it back.

Files

Screenshot.png	168 KB	11/02/2012	Vadim Nebogatov
normalize_expressions.rules	15.4 KB	11/07/2012	Vadim Nebogatov
normalize_expressions.rules	16.1 KB	11/08/2012	Vadim Nebogatov
sqlFunctions.tar.gz	2.6 KB	12/11/2012	Vadim Nebogatov
test_results.odt	16.6 KB	12/12/2012	Vadim Nebogatov
test_results.odt	17.5 KB	12/17/2012	Vadim Nebogatov
test_results.odt	19.6 KB	01/14/2013	Vadim Nebogatov
changed.tar.gz	43.1 KB	01/14/2013	Vadim Nebogatov