Base Language - Feature #7045

re-implement "normal" (non-abend) stack unwinding to avoid throwing an exception

01/16/2023 10:29 AM - Greg Shah

Status:	Test	**Start date:**	
Priority:	Urgent	**Due date:**	
Assignee:	Eduard Soltan	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**vendor_id:**	GCD

Description

Related issues:

Related to Base Language - Feature #6821: java collection performance	**WIP**
Related to Base Language - Bug #7176: cleanup p2j.oo code considering the #70...	**New**
Related to Base Language - Support #4133: reduce use of throw for control flow	**New**

History

#1 - 01/16/2023 10:40 AM - Greg Shah

- Related to Feature #6821: java collection performance added

#2 - 01/17/2023 09:24 AM - Greg Shah

I'm working on a simple approach for now. Rather than reimplementing all of the flow control for the cases where there are TM scopes, I am detecting RETURN statements in all blocks that have no TM scope or which are directly in the top-level function scope. The changes are almost ready, but I'm wondering about methods that return a value. I think this should work for them as well, they are just a variant of BlockType.FUNCTION.

Constantin: Am I forgetting something?

#3 - 01/17/2023 06:26 PM - Greg Shah

- File return_normal_java_before_and_after_20230117.zip added

In addition to user-defined functions, 2 additional callable program types which are efectively the equivalent of user-defined functions (OO property getters and non-void OO methods). I've enhanced Stanislav's testcase to add OO properties and non-void methods AND also to check the behavior of nested blocks (both with and without properties).

```
procedure proc0:
   def input parameter num as int.
end.

function func0 returns int (input num as int):
   return 0.
end.

def var flag as logical init true.

function func1 returns int (input num as int):
   if flag then
   do:
      return 0.
   end.
end.

function func2 returns int (input num as int):
```

```
   repeat:
      return 0.
   end.
end.

def var l1 as int64.
def var l2 as int64.
def var i as int.
def var num as int.
def var obj as oo.Foo.
obj = new oo.Foo().

l1 = etime.
do i = 1 to 1000000:
   run proc0(i).
end.
l2 = etime.
message "proc0 " + string(l2 - l1) + " ms".

l1 = etime.
do i = 1 to 1000000:
   func0(i).
end.
l2 = etime.
message "func0 " + string(l2 - l1) + " ms".

l1 = etime.
do i = 1 to 1000000:
   func1(i).
end.
l2 = etime.
message "func1 " + string(l2 - l1) + " ms".

l1 = etime.
do i = 1 to 1000000:
   func2(i).
end.
l2 = etime.
message "func2 " + string(l2 - l1) + " ms".

l1 = etime.
do i = 1 to 1000000:
   obj:m1(i).
end.
l2 = etime.
message "m1 " + string(l2 - l1) + " ms".

l1 = etime.
do i = 1 to 1000000:
   obj:m2(i).
end.
l2 = etime.
message "m2 " + string(l2 - l1) + " ms".

l1 = etime.
do i = 1 to 1000000:
   obj:m3(i).
end.
l2 = etime.
message "m3 " + string(l2 - l1) + " ms".

l1 = etime.
do i = 1 to 1000000:
   num = obj:test1.
end.
l2 = etime.
message "prop test1 " + string(l2 - l1) + " ms".

l1 = etime.
do i = 1 to 1000000:
   num = obj:test2.
end.
l2 = etime.
message "prop test2 " + string(l2 - l1) + " ms".
```

```
l1 = etime.
do i = 1 to 1000000:
   num = obj:test3.
end.
l2 = etime.
message "prop test3 " + string(l2 - l1) + " ms".

l1 = etime.
do i = 1 to 1000000:
   num = obj:test4.
end.
l2 = etime.
message "prop test4 " + string(l2 - l1) + " ms".
```

```
using oo.*.

class oo.Foo:

   define var flag as logical init true.

   define public property test1 as integer no-undo
      get:
         return 0.
      end get.

   define public property test2 as integer no-undo
      get:
         if flag then
         do:
            return 0.
         end.
         return 14.
      end get.

   define public property test3 as integer no-undo
      get:
         repeat:
            return 0.
         end.
      end get.

   define public property test4 as integer no-undo
      get.

   method public logical m1(input num as int):
      return false.
   end method.

   method public logical m2(input num as int):
      if flag then
      do:
         return false.
      end.
      return true.
   end method.

   method public logical m3(input num as int):
      repeat:
         return false.
      end.
   end method.
end.
```

I created branch 7045a from trunk revision 14478. I've run the above testcase in trunk rev 14479 and using 7045a revision 14479 which has a working version of this idea:

> Eliminate ReturnUnwindException usage (expensive throw/catch processing) for control flow in some cases. At this time, we will only do this for normal returns (BlockManager.returnNormal()) which are directly in a function/OO property getter/OO method AND which are not nested in an inner block that has any properties. The returnNormal() is converted to a new storeReturnValue() and a Java return is inserted if needed (only if

this is not the last statement in the function/getter/method). The runtime is changed to avoid the throw of the RUE in returnWorker().

Even though this is only a partial solution to the overall problem, it does in fact occur everywhere. I've tested Hotel GUI and the conversion changes returns all over.

I didn't see the huge difference in procedure/function difference in performance (SVL saw 25 seconds on his system). But the performance improvement is still quite clear.

Testcase	Trunk Revision 14479	7045a Revision 14479	% Improvement
proc0	1632ms	1627ms	0%
func0	1172ms	891ms	23.98%
func1	1073ms	824ms	23.21%
func2	1860ms	1857ms	0%
meth1	1267ms	906ms	28.49%
meth2	1189ms	846ms	28.85%
meth3	1959ms	1936ms	0%
prop1	1097ms	738ms	32.73%
prop2	1088ms	790ms	27.39%
prop3	1848ms	1842ms	0%
prop4	1094ms	735ms	32.82%

The cases with 0% are expected to not improve (they are the cases that are nested in blocks with properties).

The result does improve performance of these targeted function/getter/method cases. The testcases show the cost of an empty function and the improvement is significant.

I'm attaching the before/after Java results for the testcases.

#4 - 01/17/2023 06:28 PM - Greg Shah

- Status changed from New to Review

- % Done changed from 0 to 20

Constantin: Please review 7045a revision 14479.

#5 - 01/18/2023 02:33 AM - Constantin Asofiei

Greg Shah wrote:

> Constantin: Please review 7045a revision 14479.

Isn't BlockManager.returnNormal supposed to throw the ReturnUnwindException? I mean, this code should distinguish between a returnNormal and a storeReturn call, as returnNormal is still emitted in cases when it originates from an inner block:

```
      // normal return from functions will emit in converted code with an actual Java return, we can exit
      // here without throwing the exception because the Java return will cause the control flow to move
      // back to BlockManager.function() without any of the expensive throw/catch processing for the
      // ReturnUnwindException; this should work for functions, OO property getters and methods that return
      // a value
      if (rtype == ReturnType.NORMAL && wa.tm.getBlockType() == BlockType.FUNCTION)
      {
         return;
      }
```

#6 - 01/18/2023 07:39 AM - Greg Shah

> Isn't BlockManager.returnNormal supposed to throw the ReturnUnwindException? I mean, this code should distinguish between a returnNormal and a storeReturn call, as returnNormal is still emitted in cases when it originates from an inner block:

At this time, the only reason for the storeReturnValue() call is to make the converted code more readable. Anything in some nested code that has no properties will look like this:

```
if ...
{
   ...
   storeReturnValue(nnn);
   return;
}
```

instead of this:

```
if ...
{
   ...
   returnNormal(nnn);
   return;
}
```

The second form seemed confusing. Perhaps at some point we will implement different logic for the store but for now it is the same.

#7 - 01/18/2023 08:13 AM - Constantin Asofiei

My point is about this code:

```
   public integer func2(final integer _num)
   {
      integer num = TypeFactory.initInput(_num);

      return function(this, "func2", integer.class, new Block((Body) () ->
      {
         repeat("loopLabel0", new Block((Body) () ->
         {
            returnNormal(0);
         }));
      }));
   }
```

which will not exit func2 because returnNormal(0); does not raise ReturnUnwindException.

#8 - 01/18/2023 08:32 AM - Greg Shah

It does exit. I've tested it here. In this case the current block type is NOT BlockType.FUNCTION so it will throw the exception.

#9 - 01/18/2023 08:33 AM - Greg Shah

I use wa.tm.getBlockType() to check the current block instead of looking at nearest which would be BlockType.FUNCTION for all nested blocks in that top level function.

#10 - 01/18/2023 08:35 AM - Constantin Asofiei

Greg Shah wrote:

> I use wa.tm.getBlockType() to check the current block instead of looking at nearest which would be BlockType.FUNCTION for all nested blocks in that top level function.

Thanks, it makes sense now.

#11 - 01/18/2023 08:39 AM - Greg Shah

I haven't ported this to 6129b but it should apply pretty easily. Please try it on the customer app to see the difference.

#12 - 01/18/2023 02:43 PM - Greg Shah

Constantin: Do you want me to merge this into 6129b? It does require a reconversion, but I'd like to get the performance benefits into that branch.

#13 - 01/18/2023 03:24 PM - Constantin Asofiei

Greg Shah wrote:

> Constantin: Do you want me to merge this into 6129b? It does require a reconversion, but I'd like to get the performance benefits into that branch.

I was planning to wait for the conversion to finish (tomorrow morning), and after that commit this to 6129b. The 7045a branch will be merged to trunk.

#14 - 01/18/2023 03:27 PM - Greg Shah

OK, go ahead when it makes sense.

#15 - 01/19/2023 08:11 AM - Constantin Asofiei

There was a regression, in some cases like:

```
function func0 returns int.
   def var i as int.
   def var hi as handle.

   i = i + 1.
   return i.

   finally:
      delete object hi no-error.
   end.
end.
```

FWD was emitting this:

```
        return;
        silent(() ->
        {
        });
```

because of a bug in language_statements.rules.

I'm reconverting again the fix for 6129b. If this passes, I'll commit it, but 7045a needs some conversion testing before merging to trunk.

#16 - 01/19/2023 02:11 PM - Constantin Asofiei

The problem described in #7045-15 is fixed in 7045a rev 14480.

#17 - 01/19/2023 02:28 PM - Greg Shah

Code Review Task Branch 7045a Revision 14480

Weird. I don't understand the reason this works, but the code seems safe enough.

#18 - 01/20/2023 02:37 AM - Constantin Asofiei

7045a/14481 added another condition to emit a real Java return: the 4GL return statement must not have any downstream siblings, for cases like 'if true then return.', where FWD optimizes this to just 'return.'.

#19 - 01/20/2023 03:25 AM - Constantin Asofiei

The changes from 7045a are ported and committed to 6129b/14375.

#20 - 01/20/2023 08:30 AM - Greg Shah

Code Review Task Branch 7045a Revision 14481

I don't object to the change, but I'm unsure if it will work quite right without changes to convert/control_flow.rules. This code:

```
            <!-- if marked as such, emit a real Java return AFTER the block manager call, BUT ONLY IF we are
                 not the last statement (copy.parent.numRightSiblings == 0) which is also directly in the
                 function          : FUNCTION/BLOCK/STATEMENT/KW_RETURN                   or
                 OO property getter: DEFINE_PROPERTY/KW_GET/BLOCK/STATEMENT/KW_RETURN     or
                 method            : METHOD_DEF/BLOCK/STATEMENT/KW_RETURN
                 -->
            <rule>
               copy.isAnnotation("real_return") and getNoteBoolean("real_return") and
               (copy.parent.numRightSiblings != 0 or
                (copy.parent.parent.parent.type != prog.function                and
                 copy.parent.parent.parent.parent.type != prog.define_property   and
                 copy.parent.parent.parent.type != prog.method_def))
               <action>createJavaAst(java.kw_return, "", retParId)</action>
            </rule>
```

was an attempt to implement the same concept, except I didn't handle the FINALLY block case. Don't we need to handle that in this code?

FYI, I added a history entry in rev 14482.

#21 - 01/20/2023 08:34 AM - Constantin Asofiei

The code in control_flow.rules considers the case where the 4GL return. statement is the last one in the top-level block - in this case, there is no need to emit a Java return, as the block will exit anyway, naturally.

What I mention in my case is something like this:

```
function func0 returns int.
   if true then return 1.

   message "unreachable".
end.
```

which gets converted in FWD to something like this, without my change:

```
   storeReturnValue(1);
   return;
   message("unreachable").
```

This is uncompilable code in Java, as after a return; statement, there can't be any unreachable code.

So, if we find that there is code on the 'right-side' of the 4GL return. statement, then a returnNormal(1); is emitted, which will throw a ReturnUnwindException.

#22 - 01/20/2023 08:51 AM - Greg Shah

I understand. My concern is not with your change, it is with my code in convert/control_flow.rules. It doesn't consider the FINALLY case, so there are places where we will emit a return; when it is not really needed. This is just a cleanup item, it isn't functional.

#24 - 03/07/2023 04:14 AM - Constantin Asofiei

In #7142, we found that p2j.oo code still relies on returnNormal to throw a ReturnUnwindException. A fix is in 7142a which will end up in trunk.

Greg: for a permanent solution, we still need to refactor the p2j.oo code, to use the new storeReturnValue and Java return approach (and #7122 to avoid using BlockManager wrappers for plain getter/setter). But, I don't like changing the semantics of returnNormal, depending from where it is being called - as this can be prone to errors with hand-written code. Maybe we can add a new API which doesn't throw a ReturnUnwindException and gets emitted by the conversion rules?

#25 - 03/07/2023 08:03 AM - Constantin Asofiei

- Related to Bug #7176: cleanup p2j.oo code considering the #7045 and #7122 added

#26 - 03/07/2023 08:17 AM - Greg Shah

> But, I don't like changing the semantics of returnNormal, depending from where it is being called - as this can be prone to errors with hand-written code.

We can move the core logic from returnNormal into a worker and call it from both returnNormal and storeReturnValue. I don't want to duplicate logic.

#27 - 03/07/2023 08:29 AM - Constantin Asofiei

Greg Shah wrote:

> > But, I don't like changing the semantics of returnNormal, depending from where it is being called - as this can be prone to errors with hand-written code.
>
> We can move the core logic from returnNormal into a worker and call it from both returnNormal and storeReturnValue. I don't want to duplicate logic.

Yes, this can work.

Something else: a return. (or even return "something.") in an internal-procedure which is not nested in inner TM blocks, can still use the same logic as storeReturnValue.

#28 - 07/13/2023 11:56 AM - Constantin Asofiei

There is another case which needs to convert to storeReturnValue:

```
function func0 returns int.
   return 0.

   catch ex as progress.lang.error:
      return 1.
   end.
end.
```

This was committed to 7300b rev 14655

#29 - 07/14/2023 02:17 AM - Constantin Asofiei

From a large customer application, there are these numbers for a RETURN:

- from within p2j.oo code, which forces a Java throw ReturnUnwindException: {FUNCTION=493}
- from application code, uses 'storeReturnValue' or otherwise determines that a Java return is possible: {CATCH=4, FINALLY=191, FUNCTION=149863}
- from application code, FWD uses a Java throw ReturnUnwindException: {FOR_LOOP=81, FOR_BLOCK=29, CATCH=105, INTERNAL_PROC=1226, DO=335, DO_TO=105, REPEAT_WHILE=212, EXTERNAL_PROC=146}

I think we can enhance the storeReturnValue for EXTERNAL_PROC and INTERNAL_PROC, too.

#30 - 07/14/2023 01:54 PM - Greg Shah

Although there is some "low hanging fruit" left to pick, when we consider how expensive it is to use Java exceptions for flow control, I think it is well worth the time to consider a more complete solution. The core problem is that by using lambdas for nested blocks of code, there is no way to return from an enclosing lambda using only a return statement in the nested/enclosed lambda.

There is a way to do this, but there is a cost. We will have to emit extra code into the converted source. The idea is that after any nested block we will need a statement that checks state and returns if "unwinding" is needed. In order to ensure we do not add runtime overhead, this should be implemented in a way that minimizes extra calls to the TM or BM, especially ones that would have to lookup context.

Please note that this is not just a solution for RETURN processing (in which we use the ReturnUnwindException) but it can also be used for LEAVE via LeaveUnwindException.

I would hope that we can also use this for the NEXT via NextUnwindException and for RETRY processing with RetryUnwindException though that will take some extra consideration because the processing of NEXT and RETRY is already highly managed inside the TM and BM.

I don't think we need to address SilentUnwindException which is supposed to only be used for an abnormal session-level exit. Thus it is not for normal flow control and is not (or should not be) performance sensitive.

We can delete ErrorUnwindException since that class is dead code (no longer used).

#31 - 07/14/2023 02:30 PM - Constantin Asofiei

From a large app, NEXT is being called some ~2000 times and LEAVE some ~1000 times. In contrast, there are ~20k of QueryOffEndException (emitted when a query.next() has no record and needs to terminate the loop) and ~10k ErrorConditionException.

#32 - 07/14/2023 05:25 PM - Greg Shah

Constantin Asofiei wrote:

> From a large app, NEXT is being called some ~2000 times and LEAVE some ~1000 times. In contrast, there are ~20k of QueryOffEndException (emitted when a query.next() has no record and needs to terminate the loop) and ~10k ErrorConditionException.

QueryOffEndException seems like a good potential. It is only raised in 13 locations and caught in 34 spots. I think the use of exceptions was a convenience to allow this to be raised deeper in the code without having to design a mechanism to return back through all the layers between these locations and the converted code where we need to honor the off end. Eliminating tens of thousands of exception uses will make a big difference. Itts time to resolve this one.

ErrorConditionException is different as it truly is an exceptional issue rather than flow of control.

#33 - 07/20/2023 07:56 AM - Greg Shah

- Assignee changed from Greg Shah to Alexandru Lungu

I know it is late in the month to consider changes to our performance workplan. I do believe that this task has very large potential. The overhead/cost of the exception processing is mostly hidden in the JVM and cannot be assigned to specific user code locations. It represents a "blind spot" for profilers.

Considering how heavily we use QueryOffEndException, I would like you to work on eliminating its usage and moving to a model where we:

- Insert code after each persistence call in the converted code which can generate QueryOffEndException. That code would detect that an "off-end condition/event" had occurred and will return from that block. For example, in a FOR EACH where the query0.next() call would generate the QueryOffEndException, we might rewrite our converted code to be if (query0.next()) return;.
- Eliminate the generation of QueryOffEndException and rework the call tree to return back the "off-end" state as part of the normal control flow.
- Eliminate any dependencies we have on off-end listener processing and query reset processing (see TransactionManager and BlockManager for all references to QueryOffEndException). The replacement of this event processing that the query infrastructure depends upon is not as obvious to me. All I know is I want it gone. ;)

#34 - 07/20/2023 08:02 AM - Alexandru Lungu

Greg Shah wrote:

> - Insert code after each persistence call in the converted code which can generate QueryOffEndException. That code would detect that an "off-end condition/event" had occurred and will return from that block. For example, in a FOR EACH where the query0.next() call would generate the QueryOffEndException, we might rewrite our converted code to be if (query0.next()) return;.

This makes a lot of sense. This is closer to the very-tested JDBC where .next returns a boolean.

- Eliminate the generation of QueryOffEndException and rework the call tree to return back the "off-end" state as part of the normal control flow.

Make it return false until the converted code is reached + make the container block aware that the next failed. Ok!

- Eliminate any dependencies we have on off-end listener processing and query reset processing (see TransactionManager and BlockManager for all references to QueryOffEndException). The replacement of this event processing that the query infrastructure depends upon is not as obvious to me. All I know is I want it gone. ;)

I will need some time to investigate this.

#35 - 07/25/2023 07:41 AM - Greg Shah

- Status changed from Review to WIP

#36 - 07/25/2023 07:42 AM - Greg Shah

- Related to Support #4133: reduce use of throw for control flow added

#37 - 07/28/2023 08:59 AM - Alexandru Lungu

I analyzed the problem and understand most of the issue. I also have designed a solution here. However, I still have 1 single decision I am still in-doubt of, so I am looking for some advice:

1. rewrite the code as if (!q.next()) return; and still use a listener for all initialized queries inside Init, so that we can detect which queries were used for iteration. Thus, we can check at each iteration if the query reached off-end, so we avoid iterating again.
   - **Good**: we don't alter the code by a lot (just add an if conditional and a return); most of the work is done at run-time to infer the iterating query
   - **Bad**: we do extra work to infer the query using listeners at initialization stage.
2. introduce a new lambda block between Init and Body, namely Iter. This will be a boolean supplier. If this boolean supplier returns false, the query reached off-end so we stop iterating.
   - **Good**: we separate concerns. The iteration of the query is in the Init block and the body is in the Body block. Also, we don't do any work at run-time; the conversion already helps us with the detection of the off-end.
   - **Bad**: we use another lambda-function. AFAIK, lately we are trying to get rid of these. Also, such change is quite invasive into the FWD API (BlockManager).

#38 - 07/28/2023 09:37 AM - Greg Shah

> introduce a new lambda block between Init and Body, namely Iter. This will be a boolean supplier. If this boolean supplier returns false, the query reached off-end so we stop iterating.

Is the idea that this is a callback from the query to the converted code to notify that off-end has occurred?

What does the code look like that is in this lambda?

#39 - 07/28/2023 09:48 AM - Constantin Asofiei

There is another approach: in a FOR EACH block, is the navigation always done via query.next()? If yes, we can:

- add a P2JQuery.hasNext() method
- pass the P2JQuery instance to the BlockManager.forEach APIs

This avoids both the lambda and the if (!q.next()) return; change. But I don't know how we should handle FOR FIRST and FOR LAST.

#40 - 07/28/2023 09:55 AM - Constantin Asofiei

FOR FIRST and FOR LAST get converted to forBlock. I think it should work by injecting the P2JQuery instance just before the Block argument (or just before the label argument). All BlockManager.forEach* and BlockManager.forBlock* APIs will need to have this change.

#41 - 07/28/2023 11:09 AM - Alexandru Lungu

Constantin Asofiei wrote:

> There is another approach: in a FOR EACH block, is the navigation always done via query.next()? If yes, we can:
>
> - add a P2JQuery.hasNext() method
> - pass the P2JQuery instance to the BlockManager.forEach APIs
>
> This avoids both the lambda and the if (!q.next()) return; change. But I don't know how we should handle FOR FIRST and FOR LAST.

That is right. It is the cut-off solution of having Iter if we know that q.next() is always the outcome. However, I think CompoundQuery has q.iterate(), so we have some kind of conditional here. There seems to be some accumulator logic related to the difference between next and iterate, but I am not sure what it implies (see QuerryWrapper.addAccumulator javadoc).

#42 - 07/31/2023 08:35 AM - Alexandru Lungu

Greg Shah wrote:

introduce a new lambda block between Init and Body, namely Iter. This will be a boolean supplier. If this boolean supplier returns false, the query reached off-end so we stop iterating.

What does the code look like that is in this lambda?

```
forEach(TransactionType.FULL, "loopLabel0", new Block((Init) () ->
{
   query0.initialize(pt, ((String) null), null, "pt.f1 asc");
},
(Iter) () ->
   query0.next() // or iterate for compound query
(Body) () ->
{
   pt.deleteRecord();
}));
```

This was the code I was thinking about. The Iter lambda returns he result of next.

Constantin,

```
forEach(TransactionType.FULL, "loopLabel0", query0, new Block((Init) () ->
{
   query0.initialize(pt, ((String) null), null, "pt.f1 asc");
},
(Body) () ->
{
   pt.deleteRecord();
}));
```

This is basically how the code looks with your suggestion. We don't have a feedback here if we need iterate or next here. However, it looks that iterate should be equivalent to next in this specific scenario, so we can switch to "always use next" under the hood.

Conclusion: IMHO, Constantin's solution looks clean, so I will start working around it. I am starting to cut-out QueryOffEndException and adapt the conversion.

#43 - 07/31/2023 08:41 AM - Greg Shah

> Constantin's solution looks clean, so I will start working around it.

Agreed. Go ahead.

#44 - 08/01/2023 07:46 AM - Alexandru Lungu

Created 7045b.

Committed conversion changes to 7045b/rev. 14678. This is still a prototype as there is no run-time support for the changes. My conversion tests pass - the query is generated as the first FOR block parameter, except the cases where the first parameter is the transaction support. In this case, the query instance is referred through the second parameter (before label or enclosed list of unmanaged labels).

The only impediment I have now is the FOR FIRST case which converts as a RAQ. The first statement in the block is q.first(). Now that such iterating statement is removed, the block is unaware of the iteration type, as we presumed that this is always NEXT (#7045-41, #7045-42). In fact, only forBlock should be aware of the iteration type, as for any forEach it is correct to presume NEXT as the default iteration type. **Maybe we shall generate a second parameter QueryConstants.FIRST for forBlock to support this.** This also applies to LAST.

#45 - 08/01/2023 08:05 AM - Greg Shah

> Now that such iterating statement is removed, the block is unaware of the iteration type

I think I understand your point. I agree a constant for the FOR BLOCK TYPE" is appropriate. Let's avoid the term "iterating" or "iteration" here. It is confusing for the FOR **block** cases since they don't iterate.

#46 - 08/02/2023 03:55 AM - Alexandru Lungu

> There is another approach: in a FOR EACH block, is the navigation always done via query.next()? If yes, we can:
> add a P2JQuery.hasNext() method

Constantin, is this a hard constraint? I feel like making P2JQuery.next return boolean is the solution here. I agree that the classic Java iterator is based on hasNext and next, but JDBC allows only next returning boolean. I think the refactoring process would be huge if we introduce hasNext, nevertheless to mention the performance decrease. We can't easily implement hasNext without actually using next on the ResultSet. If we do so, we are basically moving the cursor using hasNext - which is not right. I am doing my implementation right now with boolean next(); let me know if you

have a feedback on this.

#47 - 08/02/2023 04:00 AM - Constantin Asofiei

Alexandru Lungu wrote:

> There is another approach: in a FOR EACH block, is the navigation always done via query.next()? If yes, we can:
> add a P2JQuery.hasNext() method

> Constantin, is this a hard constraint? I feel like making P2JQuery.next return boolean is the solution here. I agree that the classic Java iterator is based on hasNext and next, but JDBC allows only next returning boolean. I think the refactoring process would be huge if we introduce hasNext, nevertheless to mention the performance decrease. We can't easily implement hasNext without actually using next on the ResultSet. If we do so, we are basically moving the cursor using hasNext - which is not right. I am doing my implementation right now with boolean next(); let me know if you have a feedback on this.

Hm... I didn't consider this; I thought there is some way to realize that there are more records without advancing. This would mean the query.next() and query.iterate() will be moved from the converted code to BlockManager.forEach blocks, which IMO is cleaner anyway.

#48 - 08/24/2023 10:07 AM - Alexandru Lungu

- % Done changed from 20 to 50

This is an update on 7045b:

- I didn't faced any conversion issues in the meantime. Everything is good with #7045 rev. 14678 by now.
- I finished "butchering" the code by removing the usages of OffEndException. I refactored most of the queries to propagate back a boolean to reflect the off-end state.
- There were may places where the difference between looping / non-looping query was made, so I had to adopt a unified solution.
- There was a whole QueryOffEndListener logic done around init do "catch" the queries and buffers that were initialized in a for-each block. This is no longer needed as we have a reference to the query anyway.
- I've written some preliminary tests with single/multi scrolling/non-scrolling dynamic/preselect table queries. I will need to extend this further.
- I am facing some regressions early on, but I just started testing (a couple of hours ago). Most of them are related to CompoundQuery and the way it identifies if **any** of its components reached off-end (and not an outer-join component).

I will move this to 50%.

#49 - 09/07/2023 10:59 AM - Alexandru Lungu

I've continued working on 7045b. I mostly covered some issues with CompoundQuery. Currently, I am still facing some issues with CompoundQuery and FindQuery that I am addressing right now. I am still looking forward to extend the suite of test with some Presort queries and extensively test: multi-table queries with OUTER, FIRST and LAST.

EDIT: PreselectQuery and AdaptiveQuery were fully tested and there are no issues. I used OPEN QUERY and FOR EACH constructs to test these (SCROLLING and NON-SCROLLING, using FIRST, PREV, CURRENT, NEXT and LAST)

The biggest concern now is the code:

```
message "=== Preselect REPEAT ===".
do i = 1 to 2 on stop undo, leave:
    message "Try" i.
    repeat preselect each tt:
        find next tt.
        message tt.f1.
    end.
    message available(tt).
end.
```

This is generated as query.next() inside a repeat block. Currently, I modified conversion only for forEach and forBlock. I don't think it is right to include the query into the repeat block. I mean, it sounds right for forEach and forBlock as they are query related anyway. But for repeat, the PRESELECT clause is optional. That is why the technique in #7045-47 looks weird now. I am still thinking about this scenario - currently it converts, but runs infinitely.

#50 - 10/06/2023 10:37 AM - Alexandru Lungu

- Assignee changed from Alexandru Lungu to Eduard Soltan

Eduard, please focus on this task for the upcoming Sprint.

There is a lot of information here, so feel free to ask for feedback. I will send you some tests I am doing my checks on. The current implementation should be fully tested to ensure that whatever was implemented by now is right:

- Mind extending the test-cases set to ensure that what is implemented now is OK.
- Do a bzr diff to check what changes were made. There are **a lot**, so mind going through them thoroughly and check that nothing was missed. Eventually, if there is a regression, ensure that the current behavior matches the old behavior (instead of throwing QOE, a return is done).
- The only feature that is left for implementation is #7045-49. Mind using the same approach as for forEach and forBlock to the repeat block (overload with a query parameter). Mind that for repeat statement, the query is not mandatory.

No need to test with large customer applications now; we need to make sure we treat the obvious errors **now**.

#51 - 10/10/2023 03:41 AM - Eduard Soltan

There is a problem with current implementation on 7045b.

```
def temp-table tb1 field f as int field g as int.
def temp-table tb2 field f as int.
form tb1.f tb1.g tb2.f with down frame f.

def var i as int.
def var j as int no-undo.

do i = 1 to 5 transaction:
   create tb1.
   tb1.f = i.
   tb1.g = i.
end.

i = 11.
for each tb1:
   j = j + 1.
   message j.

   display tb1 with down frame f.
   down with frame f.

   tb1.f = 20.
   i = 44.

   find next tb2.
   message "found".
end.

message "i = " i.
message "tb1 avail = " available(tb1).
if available(tb1) then message "tb1.f = " tb1.f "tb1.g = " tb1.g.
pause.

clear frame f all.
for each tb1:
   display tb1 with down frame f.
   down with frame f.
end.
```

In this example tb2 is an empty temporary table. And we are iterating over tb1 with a for each block with default on endkey undo, leave. Inside this for block we call find next tb2, which should rise an endkey error in 4gl and leave the for block.
However with the current code on 7045b, for block lasts to the the last record in tb1, without leaving the for block on fisrt record.

#52 - 10/10/2023 03:49 AM - Alexandru Lungu

Eduard, 7045b is in a "protoype" stage. There may be multiple such issues with it. In fact, the (first) implementation was just finished. It requires a second iteration of the changes and testing. Please do a step by step fixing of this issues and post only a final report of what have you fixed.

This is not a usual task that attempts to fix something. Is a task that attempts to refactor code without breaking. Currently, it is still in an intermediary stage. No need to report each issue on the way, as there may be many (and related). Keep in mind that if it wasn't working before the changes, it shouldn't be fixed now asap (eventually an issue can be opened afterwards). The goal here is to have a very similar flow of things, but with RETURN TRUE|FALSE instead of QueryOffEndException.

Please work with bzr diff to check if the changes were inaccurate and attempt to fix only parts that were touched in the process of refactoring. Maybe you will need to rollback or improve the changes in some parts. For example, in #7045-51, you shall check how it was working previously (throwing QOE) and why it is not working now (returning at the right time, the right values).

#53 - 10/13/2023 10:11 AM - Eduard Soltan

During some investigation I came across QueryConstants.RETRIEVE_MODES used in CompoundQuery.

I tried to call last on open query q3 for each tt, each tt2 where tt.f1 = tt2.f2. query, however in fwd previous is called instead of last. This is due to the use of RETRIEVE_MODES matrix. I have to mention that is the above mentioned query is optimized into a join query.

Can someone provide some assistance, why QueryConstants.RETRIEVE_MODES is used?

#54 - 10/16/2023 09:08 AM - Alexandru Lungu

Eduard, please make a separate issue for #7045-53 if this can be reproduced without 7045b changes. We need to focus on having an identical solution, but without exceptions. Latent correctness concerns can be handled in other tasks.

Also, please make an update on the plan in #7045-50. What is the current status of testing? Are there other consistent changes done? (if yes, maybe prepare an intermittent commit).
In the current state of affairs, are we ready to attempt a conversion + testing on Hotel GUI for a start?

#55 - 10/16/2023 10:22 AM - Eduard Soltan

Committed on 7045b, revision 14681.

Because there are cases where an error still should be throw, I reintroduced QueryOffEnd exception. For example a find first inside a for block with on error statement.
So for now, only queries that are attached to a for block, does not throw any QueryOffEnd exceptions at all. I introduced a flag to determine if a query is attached to a for block.

I tested my changes on a few cases (for each, for first, for last), on define query construct(with scrolling and non-scrolling).
Simple multi-table joins works fine.

CompoundQuery with optimized query seems to cause quite a few issues.

#57 - 10/17/2023 10:20 AM - Eduard Soltan

Committed on 7045b, revision 14682.

For now QueryOffEnd exception is not throw in case a query is part of a for block and it exits the for block in case (query.next(), query.iterate(), query.first() or query.last()) return false;

I think we could also not throw QueryOffEnd exception, for queries defined with open query construct and also for find queries with no-error attribute.

However, I think it makes sense for find queries without no-error, to throw an exception. for, do, repeat blocks catches the error of find query inside a block and execute the on error statement.

For example, here tb2 has no element and throws a error, and leave is executed:

```
for each tb1 on error, leave:
   j = j + 1.
   message j.

   find next tb2.
   message "found".
end.
```

#58 - 10/24/2023 03:20 AM - Eduard Soltan

Committed on 7045b, revision 14683.
Also tested changes to forBlock, forEach blocks and also to next, last, first, last, unique methods logic on majic_regression_tests project. And 100% of testes pass with the current version of 7045b.

#59 - 10/24/2023 09:27 AM - Greg Shah

Eric: Please review.

#60 - 10/24/2023 09:28 AM - Greg Shah

- Status changed from WIP to Review

#61 - 10/24/2023 09:33 AM - Alexandru Lungu

> Eric: Please review.

The changes are quite extensive, so the review will take a bit :) I am converting a big application with 7045b right now and keep you updated with the results. I am also signing up for the review process once my conversion ends and there are no problems.

#62 - 10/25/2023 04:26 AM - Alexandru Lungu

- % Done changed from 50 to 70

- Status changed from Review to WIP

Eduard, I am facing some regressions with CompoundQuery.retrieveImpl (mostly NPE). I see there is a duplicate code there; maybe it needs some clean-up. Mind that retrieveImpl can return (or not) subData depending on the peek option. Also, it should either throw QOE or not depending if it is a free query (OPEN QUERY) or bound query (FOR EACH). Please do some extensive testing with CompoundQuery on the latest version you have.

In case you can't reproduce, I will dig deeper to find you a replica test-case.

#63 - 10/25/2023 04:27 AM - Alexandru Lungu

PS: good news, a large customer application converted (~17 hours) and compiled (~1 hour) correctly. Also, it passed start-up (including appservers), but it failed at run-time due to #7045-62.

#64 - 10/25/2023 09:47 AM - Alexandru Lungu

Eduard, I see you fixed this (#7045-62) in a new revision. I will need to recompile the project I use for test in order to pick up the new API definition (for current). I will keep you updated with the results with the updated version.

Please continue the work on statically reviewing the changes made in 7045b (a.k.a make a diff with all changes and try to spot any obvious problems). Even if the tests pass, we need to conceptually be sound with this solution - agree that we didn't miss some weird testcase in the process. Mind that the CompoundQuery issue could have been spotted easily from a diff.

#65 - 10/27/2023 11:10 AM - Alexandru Lungu

The NPE is fixed now, but the regression tests I have still fail. Again, I don't have a replicate (just that the expected output of the tests is not matching). I had to combine with another branch to have my tests going; hopefully it is not due to the local rebase I made.

Eduard, lets start a full iteration of the results to make sure we are not missing obvious mistakes. I will do the same thing in parallel (there are a lot of changes). If we can't find anything valuable, I will dig into the failing test.

#66 - 11/01/2023 07:29 AM - Alexandru Lungu

- Priority changed from Normal to High

I've reviewed the changes. I intend to change several things, but I will list them here for further advice:

- What is the purpose of isInsideBlock? Is this to mark that the query is used for a for iteration? I think we can remove it as part of my feedback regarding my next comment. Also, adding it as a parameter for setRecord is a bit of an overhead. Please note the widespread errorIfNull flag. We either honor it or remove it.
- I still want to get rid of QueryOffEndException for AbstractQuery implementations. I think the QOE should be thrown only on FIND scenarios; but the FIND is never implemented with AdaptiveQuery, PreselectQuery, CompoundQuery etc.. Thus, I don't think it is right to have such pattern:

```
try
{
   available = super.first(lockType);
}
catch (QueryOffEndException e)
{
   cursor.addResultFirst(null);
   throw e;
}
```

I think it is not right to have both available and QueryOffEndException. Eduard, can you clarify in which cases we need to throw QOE as an error condition to the user, beside FIND? I will like to attempt something like: never QueryOffEndException except cases where absolutely necessary **instead of** always use QueryOffEndException except cases where we iterate.

- In CompoundQuery, retrieve doesn't expect to return booleans. At best, it should return DMO or DMO ids. The code is not good and is suspect to ClassCastException.The same goes for return new Object[] { query.first(lockType) };, etc. I like the Optional attempt more, as it clearly separate: no results, has result but didn't peek, has result and peeked

```
if (rowData == null)
{
   rowData = new Object[] { true };
}
```

- We need to double-check RandomAccessQuery. I am not sure we done next and finalizeFind correctly - we return before reseting the referenceRecord and calling accumulate.

Minor changes done:

- renamed recordPresent into available in AdaptiveFind
- In AdaptiveQuery, I think it is right to do cursor.addResultFirst(getCurrentIds()) even if we didn't found a result. This will push null which marks the EOQ.
- there was a "hidden" flaw in previous - an else was missed so that there was a block that was always executing.
- Removed unused import from CompoundQuery and QueryWrapper
- I removed the ProgressiveResults fix as it had another one in trunk (which I think is more accurate).
- I removed the changes in QueryConstants

Minor review:

- Please add javadoc to isInsideBlock and history entry to AbstractQuery
- Maybe we can optimize QueryWrapper, handleQueryOffEnd lambda to avoid new logical flow

My point here is to get more aggressive with the purging of QueryOffEndException. I agree with you that we still need it (e.g. FIND FIRST tt where there is no tt). But otherwise, I can't find them a proper usage.

#67 - 11/01/2023 07:30 AM - Alexandru Lungu

P.S. please look into errorIfNull flag. This will save us from detecting cases where we need / we don't need to generate QueryOffEndException.

#68 - 11/06/2023 09:10 AM - Eduard Soltan

Alexandru Lungu wrote:

> I've reviewed the changes. I intend to change several things, but I will list them here for further advice:
>
> - What is the purpose of isInsideBlock? Is this to mark that the query is used for a for iteration? I think we can remove it as part of my feedback regarding my next comment. Also, adding it as a parameter for setRecord is a bit of an overhead. Please note the widespread errorIfNull flag. We either honor it or remove it.

I used just an auxiliar data member to know if a query was defined inside a forBlock.
To fulfill the goal of not throwing QueryOffEnd exception in more cases, I think we sholud use lenientOffEnd flag, as errorIfNull is used to throw FIND First/Last not found exception in find first/last buffer statement.

> - I still want to get rid of QueryOffEndException for AbstractQuery implementations. I think the QOE should be thrown only on FIND scenarios; but the FIND is never implemented with AdaptiveQuery, PreselectQuery, CompoundQuery etc.. Thus, I don't think it is right to have such pattern:

I completely removed QueryOffEnd from CompoundQuery.
However for AdaptiveQuery, PreselectQuery there is a catch, find next tt gets converted into an AdaptiveFind, and next method of AdativeFind calls next method of its parent class AdaptiveQuery. So I think it is not very safe for now to completly remove catch (QueryOffEnd) from AdaptiveQuery and PreselectQuery.

> I think it is not right to have both available and QueryOffEndException. Eduard, can you clarify in which cases we need to throw QOE as an error condition to the user, beside FIND? I will like to attempt something like: never QueryOffEndException except cases where absolutely necessary **instead of** always use QueryOffEndException except cases where we iterate.

I think find first/last tt are cases when error dialog appears with message FIND First/Last Faild for buffer tt.
Or find next/prev tt where just a exception is thrown.

> - In CompoundQuery, retrieve doesn't expect to return booleans. At best, it should return DMO or DMO ids. The code is not good and is suspect to ClassCastException.The same goes for return new Object[] { query.first(lockType) };, etc. I like the Optional attempt more, as it clearly separate: no results, has result but didn't peek, has result and peeked

Change to used return query.getRow() for queries with peek = false.

Committed the changes on 7045b, revision 14686.

#69 - 11/08/2023 09:25 AM - Alexandru Lungu

Eduard, mind setting up some large customer applications and do some smoke tests. I will take the changes and do on some other applications I have set-up. I am planning to do a review first thing tomorrow morning.

#70 - 11/09/2023 03:59 AM - Alexandru Lungu

I rebased 7045b to latest trunk. It is now at rev. 14833.
Eduard, please double-check: RecordBuffer (especially setRecord), PreselectQuery and AdativeQuery (especially coreFetch).

EDIT: I will need to reconvert my testing application now that 7045b was rebased.

#71 - 11/15/2023 03:40 AM - Eduard Soltan

Converted and setup a large customer application using 7045b branch, and used it for a bit. Did not noticed any major issues, like application crash or noticeable regressions.

Also committed on 7045b, revision 14834.

#72 - 11/15/2023 07:31 AM - Greg Shah

If you think it is ready, let's get ChUI regression tests, ETF and the big POC tested.

#73 - 11/15/2023 08:09 AM - Alexandru Lungu

Greg, I can do that for POC, but it will take me a day to have it converted. Also mind that I am already reconverting with #6649, so this will be delayed for another day :/

#74 - 11/15/2023 08:12 AM - Greg Shah

Understood. For such extensive changes to our control flow, it seems necessary.

#75 - 11/21/2023 07:52 AM - Alexandru Lungu

Eduard, I didn't had the chance to convert yet the POC. Can you attempt it? I have already queued some other performance items, so I missed the timing. Thank you!

#76 - 11/21/2023 02:40 PM - Greg Shah

Eric: Please don't forget to do a code review ASAP.

#77 - 11/24/2023 04:11 AM - Eric Faulhaber

Code review 7045b/14825-14834:

There are file header entries problems (i.e., duplicate entries in some files, missing entries in most files). For example, control_flow.rules entries 99 and 101; database_access.rules entries 122 and 125; BlockManager entries 62 and 69; etc. Please fix these. AdaptiveFind and other classes have edits but no file header entries. This list is not exhaustive. Every file that changed must have a header entry.

The javadoc comments for every query navigation method which has changed its control flow mechanics (from exceptions to boolean return value) needs to be updated to explain the new control flow. Accordingly, if QueryOffEndException is no longer thrown by these methods, there should no longer be a throws javadoc tag for that exception. Check whether class-level javadoc should be updated as well.

I have noted the discussion above that QOEE is not yet fully eliminated. I see this, for instance, with the try-catch blocks in AdaptiveQuery.first(LockType). Do we have an idea how much effort is left to fully achieve the removal of QOEE? I ask because as long as we are in this hybrid state of implementing error handling both using exceptions and return values, the code is more complex and we are asking for regressions.

P2JQuery.setLenientOffEnd(boolean) (and every override of it; e.g., QueryWrapper.setLenientOffEnd(boolean)) needs javadoc. This is especially important, since the semantic of "lenient-off-end" certainly has changed with this branch.

Why was the outer-join-specific logic starting around line 6206 of PreselectQuery changed/removed? I'm not saying it's incorrect, but I don't see an explanation anywhere. As in other places, QOEE is still caught and processed here. Is it actually thrown or does this error handling need to be

refactored?

QueryConstants.ITERATE needs javadoc.

RecordBuffer.setRecord is missing javadoc for the new lenientOffEnd parameter. Should this method still be throwing QOEE?

Like the query changes, the changes to the BlockManager APIs are missing javadoc for the new parameters and an explanation of the new flow control. We really should not have logic inside BlockManager which is conditional upon query instanceof CompoundQuery. Can this be avoided and refactored into the persist package without breaking the fundamental changes we are trying to achieve with this task?

This set of changes will need considerable conversion and runtime testing.

#78 - 11/29/2023 08:35 AM - Alexandru Lungu

Eduard, please make an update on your status of running the POC with your changes and addressing Eric's #7045-77.

#79 - 11/29/2023 08:59 AM - Eduard Soltan

I am addressing some regression on running POC. But I think, I am close to find the problem causing this regression. I want to make sure that I do not face any regression, and after that I will address #7045-77.

#80 - 12/04/2023 03:07 AM - Eduard Soltan

Eric Faulhaber wrote:

> I have noted the discussion above that QOEE is not yet fully eliminated. I see this, for instance, with the try-catch blocks in AdaptiveQuery.first(LockType). Do we have an idea how much effort is left to fully achieve the removal of QOEE? I ask because as long as we are in this hybrid state of implementing error handling both using exceptions and return values, the code is more complex and we are asking for regressions.

Well, I think that QueryOffEnd can not be fully eliminated as there are cases when this kind exception is thrown in 4gl, and from that depends to control flow of program.
For example in case of an do block with an on error undo statement, if there is an find next tt statement inside that just returns a boolean, it will change the control flow of the program.
Basically it will iterate while the do block condition is met, when it should exit the block at the moment find next tt returned no result.
And this statement is converted in AdaptiveFind query, we can get rid of catching QQE in its parent classes AdaptiveQuery and PreselectQuery

> Why was the outer-join-specific logic starting around line 6206 of PreselectQuery changed/removed? I'm not saying it's incorrect, but I don't see an explanation anywhere. As in other places, QOEE is still caught and processed here. Is it actually thrown or does this error handling need to be refactored?

Sorry, this was probably removed at rebase.

> RecordBuffer.setRecord is missing javadoc for the new lenientOffEnd parameter. Should this method still be throwing QOEE?

As I said previously, there are cases where it is natural to throw such an exception, I added lenientOffEnd paramenter to check cases when this error should not be thrown. (queries define inside an for each block or using define query statement).

> Like the query changes, the changes to the BlockManager APIs are missing javadoc for the new parameters and an explanation of the new flow control. We really should not have logic inside BlockManager which is conditional upon query instanceof CompoundQuery. Can this be avoided and refactored into the persist package without breaking the fundamental changes we are trying to achieve with this task?

I think this could be done easily, because iterate method of CompundQuery calls retrieveImpl with direction parameter as NEXT. So I could just leave call to query.next() inside the BlockManger.

I finally managed to run POC, converted using 7045b branch without any regressions.

#81 - 12/04/2023 03:47 AM - Alexandru Lungu

Eduard, please address the review and commit a final solution to 7045b (including javadoc / history entries / etc.) and ensure it is working with POC. Afterwards, move it to 100% Done and Internal Test.

I would like to have it profiled asap and eventually merge. Can you have it done in the next couple of hours, so I have time to complete the profiling today?

Also mind my last concern: can you please test uast/query_off_end/qoe_exception?
I am also interested in the following constructs:

- [redacted]
```
   message tt.f1.
end.
```
- [redacted]
```
   message tt.f1.
end.
```
- [redacted]
```
   message tt.f1.
end.
```

#82 - 12/04/2023 05:54 AM - Eduard Soltan

Committed on 7045b, revision 14835.

I tested uast/query_off_end/qoe_exception , did not see any regressions. In all the cases that you mentioned query is just preselected, and an iteration is performed only programmer explicitly call an operation on that query (first, last, etc). And when query.next() does not have any record to fetch, QQE is raised in 4gl and also in fwd.

#83 - 12/04/2023 06:41 AM - Eduard Soltan

- % Done changed from 70 to 100

#84 - 12/04/2023 10:38 AM - Alexandru Lungu

- Status changed from WIP to Internal Test

I've run 7045b with the performance tests and got 8.168s. Comparing to the baseline of 7.958s, this is +2.5% slower. IMHO, last week I profiled ~5 different branches and got results from +1% to +3% on all "theoretically" faster solutions. I've rerun the baseline and still had that ~7.950s timing. I am concerned that we are missing a point here **or** my system is not doing great at all.

Talking strictly about 7045b, Eduard, we need some kind of statistic to see:

- how many QOE are still thrown in 7045b vs 7156b; did we had a massive reduction of such exceptions being thrown?
- further, we may need a deeper understanding on:
  - how many for-each blocks were executed and the average number of steps
  - how many for-each blocks were ended by a next returning false
  - the different query flavors: PreselectQuery vs AdaptiveQuery vs PresortQurery vs CompoundQuery used in for-each blocks.

I will review the changes to spot any obvious performance downgrade.

#85 - 12/05/2023 11:01 AM - Alexandru Lungu

I am looking now in the changes of 7045b:

- Eduard, mind that you have some hard-tabs in your last commit (AdaptiveFind history entry and in BlockManager where you removed the iterate).
- I think the outer logic is still lost and should be added back (in PreselectQuery). This won't be noticeable in the POC tests, as 7156b disables outer-joins for PreselectQuery.
- You have places where you set referenceRecord and call accumulate **only** if the DMO is not null. This is not compatible with the original code (pre-7045b). For instance, in RandomAccessQuery.next, the "old" code was doing:

```
OffEnd offEnd = (dmo == null ? OffEnd.BACK : OffEnd.NONE);
updateBuffer(dmo, lockType, false, offEnd);
referenceRecord = dmo;
```

- addResultLast and addResultFirst changes are right from my POV.

I retested performance today and it is still "slower" side.

#86 - 12/06/2023 02:00 AM - Eduard Soltan

Alexandru Lungu wrote:

> I've run 7045b with the performance tests and got 8.168s. Comparing to the baseline of 7.958s, this is +2.5% slower. IMHO, last week I profiled ~5 different branches and got results from +1% to +3% on all "theoretically" faster solutions. I've rerun the baseline and still had that ~7.950s timing. I am concerned that we are missing a point here **or** my system is not doing great at all.
>
> Talking strictly about 7045b, Eduard, we need some kind of statistic to see:
>
> - how many QOE are still thrown in 7045b vs 7156b; did we had a massive reduction of such exceptions being thrown?
> - further, we may need a deeper understanding on:

- how many for-each blocks were executed and the average number of steps
- how many for-each blocks were ended by a next returning false
- the different query flavors: PreselectQuery vs AdaptiveQuery vs PresortQurery vs CompoundQuery used in for-each blocks.

I will review the changes to spot any obvious performance downgrade.

This is pretty strange, I tested POC with 7156 and POC using 7045b conversion, and this are some results:
- POC with 7156b, warmup and 1 test, around 40000 times QQE is thrown.
- POC with 7045b conversion, also warmup and 1 test, 1021 QQE exceptions are thrown.

I also made quite a few profilings yesterday, and this are some results:
- POC with 7156b, the most time consuming method is RecordBuffer.throwOffEnd, it takes around ~12% of time of execution of 20 performance tests.
- POC with 7045b, RecordBuffer.throwOffEnd takes considerable less time, around ~1% of the time.

Also every time I tested POC with 7045b was with at least 1s faster.

#87 - 12/06/2023 06:18 AM - Alexandru Lungu

Eduard, please focus on reaching a "state-of-the-art" solution for 7045b (also referring to #7045-84). Also, triple check we didn't miss something from it.
I intend to start a (very) long running testing process with 7045b, so we need to make sure it is fully completed and the final solution tested with whatever we have available (POC, large cust. application, Majic).

#88 - 12/07/2023 08:04 AM - Constantin Asofiei

Alexandru, I haven't looked at the full code, but is the QueryWrapper.handleQueryOffEnd still needed?

#89 - 12/07/2023 08:34 AM - Eduard Soltan

I think is not needed, as QueryWrapper is used in conversion of open query statement. And in this case QQE is not thrown.
I tested on POC without QueryWrapper.handleQueryOffEnd, and got no visible regressions.

#90 - 12/07/2023 08:40 AM - Alexandru Lungu

Eduard, I started converting a really large application with 7045b (on top of 7156b). You have 3 days left to do the changes, before the run-time testing starts :)

Regarding your last comment: it depends upon the delegate right? If the delegate is not lenient-off-end, it **may** throw QOE, right? So we should ensure first that the delegates are always lenient-off-end.
Please check QueryWrapper.assign(RandomAccessQuery) - it is automatically making the query lenient-off-end. I think this can be done to the other assigned queries as well (PreselectQuery and CompoundQuery).

#91 - 12/07/2023 08:53 AM - Eduard Soltan

Alexandru Lungu wrote:

> Regarding your last comment: it depends upon the delegate right? If the delegate is not lenient-off-end, it **may** throw QOE, right? So we should ensure first that the delegates are always lenient-off-end.
> Please check QueryWrapper.assign(RandomAccessQuery) - it is automatically making the query lenient-off-end. I think this can be done to the other assigned queries as well (PreselectQuery and CompoundQuery).

In CompoundQuery, lenient-off-end is set in constructor and in retrieveImpl for every query component.
In PreselectQuery, lenient-off-end is set in open method, I quess that this method should always be called when a PreselectQuery is used as a delegate to a QueryWrapper.
But I could add setting QQE in QueryWrapper

#92 - 12/12/2023 02:58 AM - Alexandru Lungu

- Priority changed from High to Urgent

Eduard, please address the latest suggestions and commit to 7045b.

Lets redo the whole testing process with this latest 7045b you have and attempt to merge:

- ChUI regression tests
- POC profiling (should be a clear improvement)
- large customer application smoke tests

I am still attempting to fix another suite of regression tests for this.
Constantin, can you also run ETF with 7045b when ready?

#93 - 12/12/2023 03:55 AM - Eduard Soltan

- File poc_converted_2.png added

- File poc_original_snapshot.png added

- File poc_original_3.png added

- File poc_original_1.png added

- File poc_converted_snapshot.png added

- File poc_converted_3.png added

Committed on 7045b, revision 14836.
- removed handleQueryOffEnd method from QueryWrapper and all of its usage.
- in RandomAccessQuery change to call accumulate even when query failed to retrieve any results.

Also run some smoke test on a large application, it seems good.

POC profiling:
This are some result of profiling on running POC tests for 20 times.

Result for running POC with clean 7156b:

Profiler Snapshot

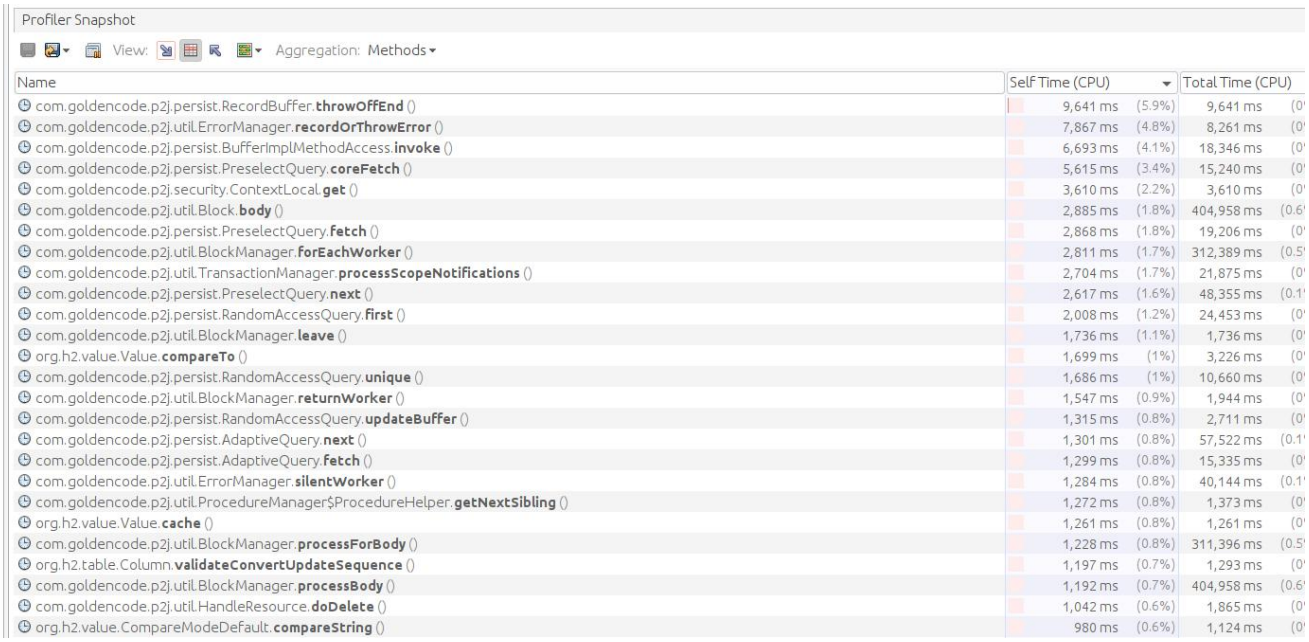
View: Aggregation: Methods

Name	Self Time (CPU)	Total Time (CPU)
com.goldencode.p2j.persist.RecordBuffer.**throwOffEnd** ()	10,622 ms (7.1%)	10,622 ms (0
com.goldencode.p2j.util.ErrorManager.**recordOrThrowError** ()	7,316 ms (4.9%)	8,271 ms (0
com.goldencode.p2j.persist.BufferImplMethodAccess.**invoke** ()	6,634 ms (4.5%)	18,124 ms (0
com.goldencode.p2j.persist.PreselectQuery.**coreFetch** ()	4,932 ms (3.3%)	14,822 ms (0
com.goldencode.p2j.util.Block.**body** ()	3,820 ms (2.6%)	356,114 ms (0.6
com.goldencode.p2j.persist.RandomAccessQuery.**unique** ()	3,108 ms (2.1%)	10,822 ms (0
com.goldencode.p2j.security.ContextLocal.**get** ()	3,090 ms (2.1%)	3,189 ms (0
com.goldencode.p2j.util.BlockManager.**forEachWorker** ()	2,758 ms (1.9%)	273,239 ms (0.5
com.goldencode.p2j.persist.PreselectQuery.**next** ()	2,758 ms (1.9%)	41,065 ms (0.1
com.goldencode.p2j.persist.RandomAccessQuery.**first** ()	2,299 ms (1.5%)	24,448 ms (0
com.goldencode.p2j.persist.PreselectQuery.**fetch** ()	2,206 ms (1.5%)	18,119 ms (0
org.h2.value.Value.**compareTo** ()	1,826 ms (1.2%)	2,638 ms (0
com.goldencode.p2j.util.TransactionManager.**processScopeNotifications** ()	1,712 ms (1.2%)	20,271 ms (0
com.goldencode.p2j.util.BlockManager.**leave** ()	1,701 ms (1.1%)	1,701 ms (0
com.goldencode.p2j.util.BlockManager.**returnWorker** ()	1,477 ms (1%)	1,577 ms (0
com.goldencode.p2j.util.BlockManager.**processForBody** ()	1,304 ms (0.9%)	273,044 ms (0.5
com.goldencode.p2j.persist.AdaptiveQuery.**next** ()	1,298 ms (0.9%)	49,976 ms (0.1
com.goldencode.p2j.util.BlockManager.**next** ()	1,297 ms (0.9%)	1,297 ms (0
com.goldencode.p2j.persist.AdaptiveQuery.**fetch** ()	1,265 ms (0.9%)	14,608 ms (0
com.goldencode.p2j.persist.orm.TempTableDataSourceProvider$DataSourceImpl$TempProxyConnection.**prepareStatement** ()	1,200 ms (0.8%)	1,402 ms (0
com.goldencode.p2j.persist.RandomAccessQuery.**updateBuffer** ()	1,102 ms (0.7%)	3,104 ms (0
org.h2.value.Value.**cache** ()	1,046 ms (0.7%)	1,046 ms (0
com.goldencode.p2j.util.ProcedureManager.**delete** ()	960 ms (0.6%)	14,082 ms (0
com.goldencode.p2j.persist.RandomAccessQuery.**lambda$unique$1** ()	893 ms (0.6%)	5,216 ms (0
com.goldencode.p2j.util.HandleChain.**delete** ()	882 ms (0.6%)	9,871 ms (0
org.apache.commons.collections4.map.AbstractHashedMap.**put** ()	804 ms (0.5%)	1,005 ms (0

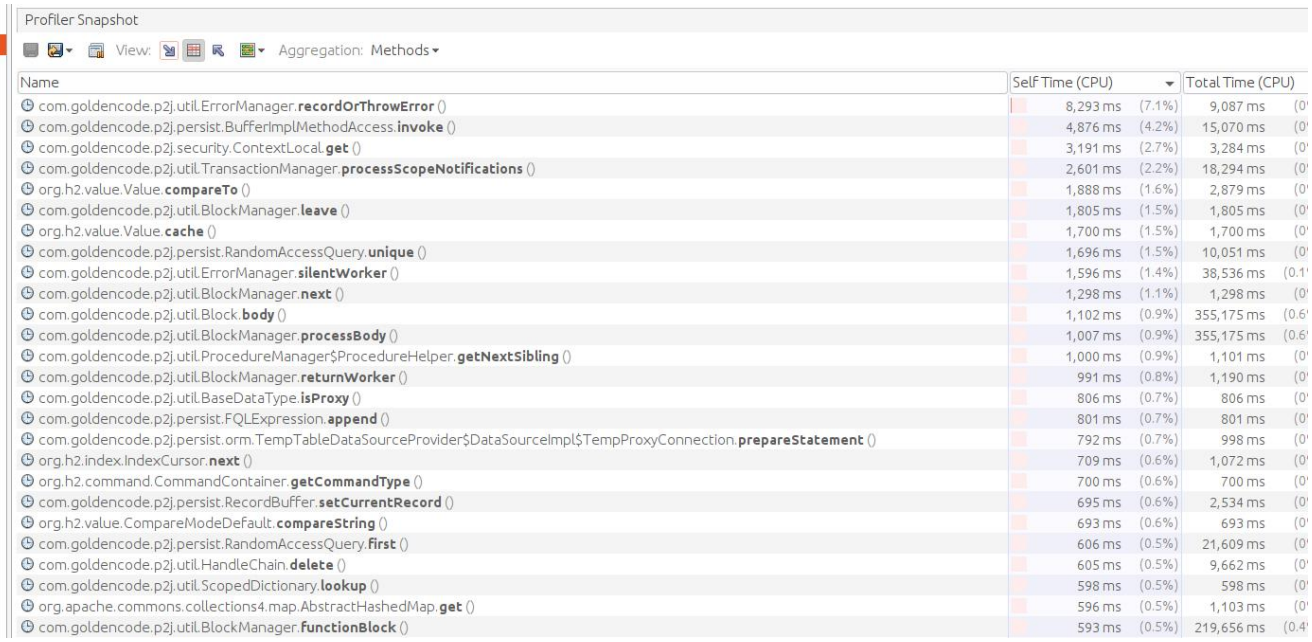
Profiler Snapshot

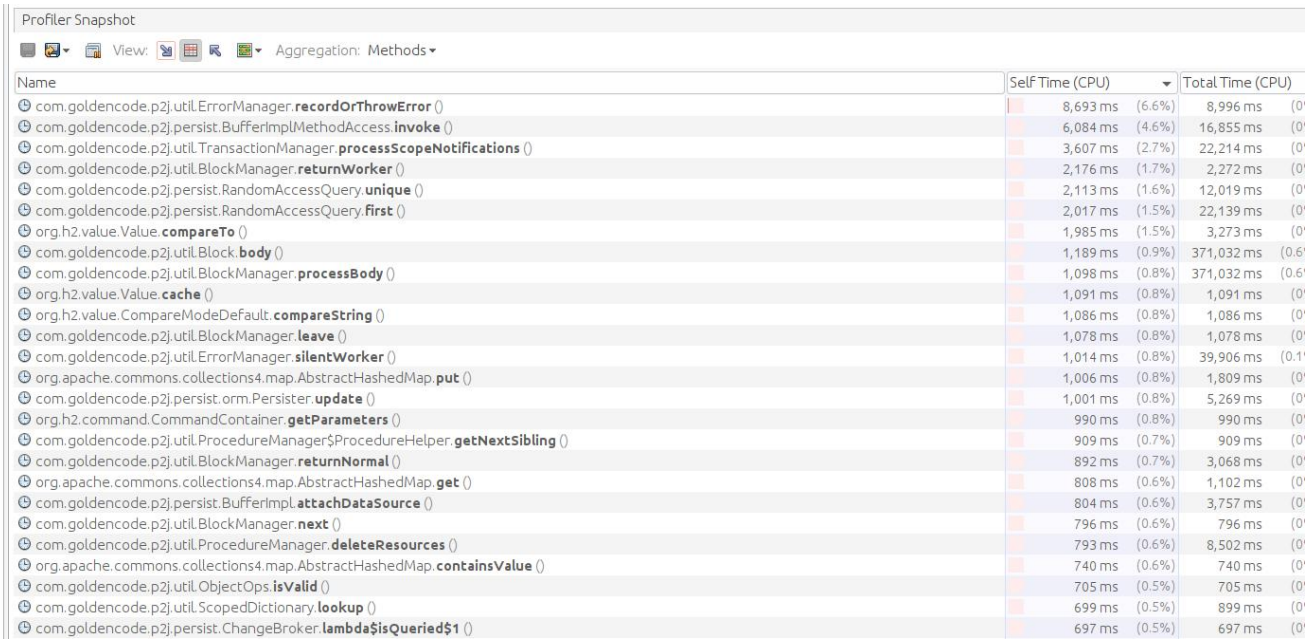
View: Aggregation: Methods

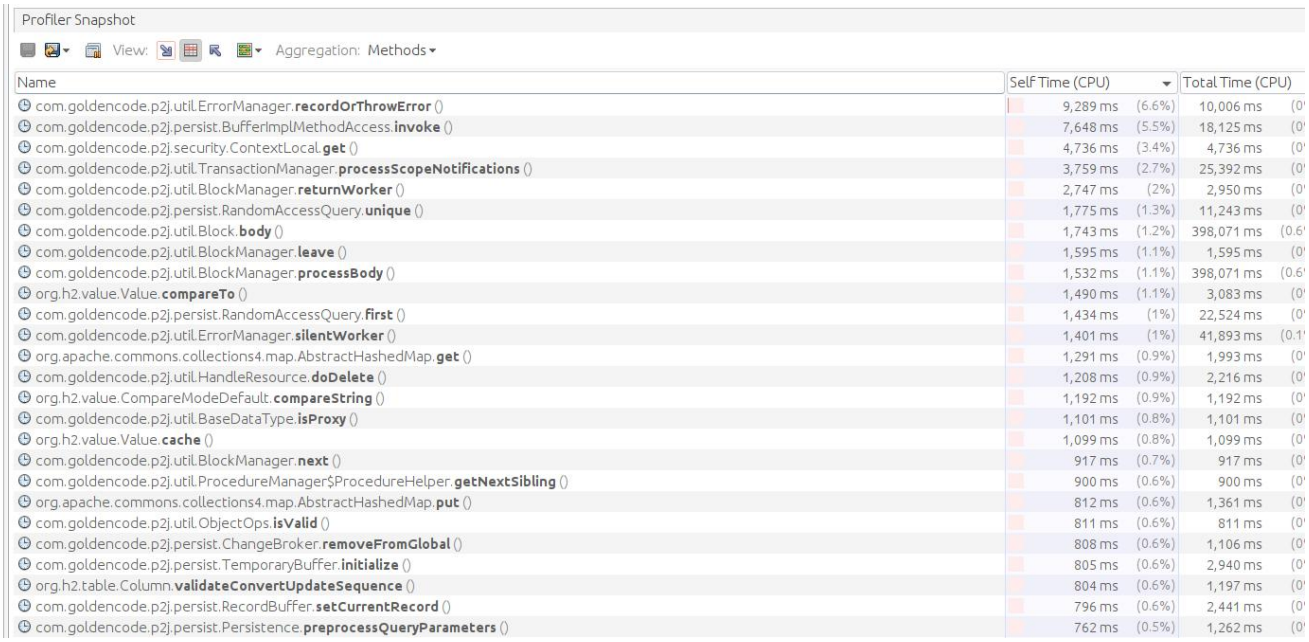
Name	Self Time (CPU)	Total Time (CPU)
com.goldencode.p2j.persist.RecordBuffer.**throwOffEnd** ()	12,684 ms (7.5%)	12,684 ms (0
com.goldencode.p2j.util.ErrorManager.**recordOrThrowError** ()	7,736 ms (4.6%)	8,334 ms (0
com.goldencode.p2j.persist.BufferImplMethodAccess.**invoke** ()	5,660 ms (3.3%)	18,309 ms (0
com.goldencode.p2j.persist.PreselectQuery.**coreFetch** ()	5,634 ms (3.3%)	17,971 ms (0
com.goldencode.p2j.util.Block.**body** ()	3,716 ms (2.2%)	413,297 ms (0.6
com.goldencode.p2j.security.ContextLocal.**get** ()	3,713 ms (2.2%)	3,814 ms (0
com.goldencode.p2j.persist.RandomAccessQuery.**unique** ()	3,001 ms (1.8%)	12,175 ms (0
com.goldencode.p2j.util.TransactionManager.**processScopeNotifications** ()	2,432 ms (1.4%)	21,917 ms (0
com.goldencode.p2j.persist.PreselectQuery.**next** ()	2,366 ms (1.4%)	51,557 ms (0.1
com.goldencode.p2j.util.BlockManager.**forEachWorker** ()	2,347 ms (1.4%)	317,106 ms (0.5
com.goldencode.p2j.persist.PreselectQuery.**fetch** ()	2,333 ms (1.4%)	21,021 ms (0
com.goldencode.p2j.util.BlockManager.**processForBody** ()	2,007 ms (1.2%)	316,449 ms (0.5
org.h2.value.Value.**cache** ()	1,896 ms (1.1%)	1,896 ms (0
com.goldencode.p2j.persist.AdaptiveQuery.**fetch** ()	1,806 ms (1.1%)	17,538 ms (0
com.goldencode.p2j.util.BlockManager.**leave** ()	1,700 ms (1%)	1,700 ms (0
com.goldencode.p2j.util.ErrorManager.**silentWorker** ()	1,620 ms (1%)	43,198 ms (0.1
org.h2.value.Value.**compareTo** ()	1,532 ms (0.9%)	2,841 ms (0
com.goldencode.p2j.persist.orm.Persister.**update** ()	1,350 ms (0.8%)	6,552 ms (0
com.goldencode.p2j.util.BlockManager.**returnWorker** ()	1,307 ms (0.8%)	1,606 ms (0
com.goldencode.p2j.persist.RandomAccessQuery.**updateBuffer** ()	1,295 ms (0.8%)	4,147 ms (0
com.goldencode.p2j.util.BaseDataType.**isProxy** ()	1,108 ms (0.7%)	1,108 ms (0
com.goldencode.p2j.uast.ProgressParser.**compare_expr** ()	1,096 ms (0.6%)	2,109 ms (0
org.apache.commons.collections4.map.AbstractHashedMap.**put** ()	1,086 ms (0.6%)	1,086 ms (0
com.goldencode.p2j.persist.orm.TempTableDataSourceProvider$DataSourceImpl$TempProxyConnection.**prepareStatement** ()	1,054 ms (0.6%)	1,346 ms (0
com.goldencode.p2j.util.ObjectOps.**isValid** ()	971 ms (0.6%)	1,073 ms (0
com.goldencode.p2j.persist.AdaptiveQuery.**next** ()	906 ms (0.5%)	61,349 ms (0.1



Result for running POC with 7045b:







Profiler Snapshot

View: Aggregation: Methods

Name	Self Time (CPU)	Total Time (CPU)
com.goldencode.p2j.util.ErrorManager.**recordOrThrowError** ()	9,289 ms (6.6%)	10,006 ms
com.goldencode.p2j.persist.BufferImplMethodAccess.**invoke** ()	7,648 ms (5.5%)	18,125 ms
com.goldencode.p2j.security.ContextLocal.**get** ()	4,736 ms (3.4%)	4,736 ms
com.goldencode.p2j.util.TransactionManager.**processScopeNotifications** ()	3,759 ms (2.7%)	25,392 ms
com.goldencode.p2j.util.BlockManager.**returnWorker** ()	2,747 ms (2%)	2,950 ms
com.goldencode.p2j.persist.RandomAccessQuery.**unique** ()	1,775 ms (1.3%)	11,243 ms
com.goldencode.p2j.util.Block.**body** ()	1,743 ms (1.2%)	398,071 ms
com.goldencode.p2j.util.BlockManager.**leave** ()	1,595 ms (1.1%)	1,595 ms
com.goldencode.p2j.util.BlockManager.**processBody** ()	1,532 ms (1.1%)	398,071 ms
org.h2.value.Value.**compareTo** ()	1,490 ms (1.1%)	3,083 ms
com.goldencode.p2j.persist.RandomAccessQuery.**first** ()	1,434 ms (1%)	22,524 ms
com.goldencode.p2j.util.ErrorManager.**silentWorker** ()	1,401 ms (1%)	41,893 ms
org.apache.commons.collections4.map.AbstractHashedMap.**get** ()	1,291 ms (0.9%)	1,993 ms
com.goldencode.p2j.util.HandleResource.**doDelete** ()	1,208 ms (0.9%)	2,216 ms
org.h2.value.CompareModeDefault.**compareString** ()	1,192 ms (0.9%)	1,192 ms
com.goldencode.p2j.util.BaseDataType.**isProxy** ()	1,101 ms (0.8%)	1,101 ms
org.h2.value.Value.**cache** ()	1,099 ms (0.8%)	1,099 ms
com.goldencode.p2j.util.BlockManager.**next** ()	917 ms (0.7%)	917 ms
com.goldencode.p2j.util.ProcedureManager$ProcedureHelper.**getNextSibling** ()	900 ms (0.6%)	900 ms
org.apache.commons.collections4.map.AbstractHashedMap.**put** ()	812 ms (0.6%)	1,361 ms
com.goldencode.p2j.util.ObjectOps.**isValid** ()	811 ms (0.6%)	811 ms
com.goldencode.p2j.persist.ChangeBroker.**removeFromGlobal** ()	808 ms (0.6%)	1,106 ms
com.goldencode.p2j.persist.TemporaryBuffer.**initialize** ()	805 ms (0.6%)	2,940 ms
org.h2.table.Column.**validateConvertUpdateSequence** ()	804 ms (0.6%)	1,197 ms
com.goldencode.p2j.persist.RecordBuffer.**setCurrentRecord** ()	796 ms (0.6%)	2,441 ms
com.goldencode.p2j.persist.Persistence.**preprocessQueryParameters** ()	762 ms (0.5%)	1,262 ms

#94 - 12/12/2023 11:07 AM - Alexandru Lungu

Eduard, there is an API issue we have on BlockManager. I think the rewriting of the BlockManager API to include the query had regressed at some point. Please double check that. This is an example which currently fails:

FOR tt WHERE tt.f1 = 2 NO-LOCK. END.

Mind that this is not FOR EACH, neither FOR FIRST. It should convert to a forBlock. This is how it was converted now:

```
forBlock(query25, "blockLabel28", new Block((Init) () -> ...);
```

Before 7045b:

```
forBlock("blockLabel28", new Block((Init) () -> ...);
```

Before 7045b we had forBlock(String, Block) method in BlockManager.

In 7045b, we don't have forBlock(P2JQuery, String, Block)!

Thus, please recheck the BlockManager API and fix the problems you encounter. Make this your top-priority. I can't run the regression tests as they don't compile.

#95 - 12/13/2023 08:27 AM - Alexandru Lungu

I discussed with Eduard and it seems that the problem is that we have forBlock(P2JQuery, int, String, Block), but we missed generating the direction, so we end up with forBlock(P2JQuery, String, Block). Thus, the conversion is broken.

I've implemented a work-around locally to have it compile, so I can get it running with the regression tests. However, the patch is quite "dirty" (adding back the old signatures and mapping them at run-time to the new signatures from BlockManager). I hard-coded the direction as UNIQUE when not specified. This isn't "bad", but it is clearly better to fix this at conversion to go the proper way.

I think the conversion changes to fix this will be quite safe and the occurrences of such constructs are very rare anyway. Run-time changes aren't required if we approach to fix this at conversion.

To have it tested "cleanly", I will need to reconvert and recompile (but this may take ~3/4 days). If I stick to the run-time testing now, I hope we can get some results first thing tomorrow morning (apparently, the compiling takes ~1day).

#96 - 12/13/2023 09:38 AM - Eduard Soltan

Committed on 7045b, revision 14837.

This change requires reconversion. 4gl constructs like for tt, should be converted in forBlock(query, QueryConstants.UNIQUE, ...).
Also made a change in BlockManager, to call query.unique() when direction parameter is equal to QueryConstants.UNIQUE.

I tested conversion on Hotel, but I also am planning today to run conversion on a big customer application.

#97 - 12/18/2023 06:29 AM - Alexandru Lungu

Eduard, please mind that there are still some hard-tabs in the last 2 commits. (BlockManager).

#98 - 12/19/2023 10:18 AM - Alexandru Lungu

7045b passes a large set of regression tests from a customer application. I tested without the changes in rev. 14837.
I was delayed a bit by some false negative tests that were due to a wrong rebase. There were 27/43 tests failing with that wrong rebase.

I retested now with everything in order (good job Eduard in catching that rebase mistake). I have 20/43 tests failing after fixing the problem. With the a baseline FWD (which I am not sure of the version), there are also 20/43 tests failing (the same ones).

However, I've done this tests with a rebased 7045b from 7156b, but before rev. 14837. I will add 14837 as a patch and retest to ensure that the latest version is OK.

Eduard, my testing is getting close to an end. Please let me know if you have something else to do on 7045b. Otherwise, we can consider it ready for merge.

#99 - 12/20/2023 09:25 AM - Eduard Soltan

Removed hard tabs from the previous commit. Committed on 7045b, revision 14838.
I do not think there is anything else to add on this branch.

#100 - 12/20/2023 09:48 AM - Alexandru Lungu

Completely finished the testing of 7045b. Pending for merge.

#101 - 12/20/2023 10:36 AM - Eric Faulhaber

Quick final look at 7045b/14838...

It looks like we still need a rebase to current trunk before merge. Also, some minor stuff to be fixed:

Header entry 122 was removed from database_access.rules. It was redundant with 125, but now the numbers are off.

There are still hard tabs in AdaptiveFind (line 106).

P2JQuery header entry 66 should not be numbered; it is in the same branch to be merged to trunk as entry 65. Same for QueryWrapper header entry 99; same branch as 98. Same for RandomAccessQuery entries 129 & 130. BlockManager: 70-72.

Where specific file header entry numbers are mentioned above, these numbers may of course change with the rebase.

As long as the rebase doesn't introduce any complications, none of the above changes should invalidate the testing done, so after this cleanup, we should still be ok for merge.

#102 - 12/20/2023 11:38 AM - Greg Shah

- Status changed from Internal Test to Merge Pending

Please merge to trunk after 8007a.

#103 - 12/20/2023 11:44 AM - Alexandru Lungu

Just rebased 7045b to latest trunk, but the process was a bit slow due to the large number of changes to be introduced. Eduard, please do some final quick tests with this branch before merging.

#104 - 12/20/2023 06:55 PM - Greg Shah

Hold off on the merge. 8056a and 8127a need to go first.

#105 - 12/21/2023 06:17 AM - Eduard Soltan

Looked at 7045b after rebase, it looks fine.
Done some minor fixes at history entry.
In forBlockWorker, changed in case of direction parameter is equal QueryConstans.unique to call query.unique instead of last.
Also, comp.getQuery().setLenientOffEnd(true) in CompoundQuery.retrieveImpl was missing after rebase.

Committed on 7045b, revision 14903.

#106 - 12/21/2023 07:15 AM - Alexandru Lungu

Nice catches, Eduard! Please do a second iteration of the changes - I really want to make sure we don't missed something here :)

Constantin, ETF tests are the only ones that were not run. I rebased 7045b to latest trunk and it is now at 14907. Eduard, before your final review,

please do an update.

#107 - 12/21/2023 09:03 AM - Constantin Asofiei

Alexandru Lungu wrote:

> Constantin, ETF tests are the only ones that were not run. I rebased 7045b to latest trunk and it is now at 14907. Eduard, before your final review, please do an update.

I'll have ETF testing later tonight, need to reconvert.

#108 - 12/21/2023 09:30 AM - Alexandru Lungu

Constantin, do you feel like it is better to get **everything** in trunk and then run ETF against all, instead of "micro" testing?

I was looking at Greg's recent email.

#109 - 12/21/2023 02:27 PM - Constantin Asofiei

7045b passed ETF testing.

#110 - 12/21/2023 02:33 PM - Greg Shah

You can merge 7045b to trunk after 3303a.

#111 - 12/21/2023 03:07 PM - Greg Shah

7045b is at the front of the queue now, go ahead and merge.

#112 - 12/22/2023 04:24 AM - Alexandru Lungu

- Status changed from Merge Pending to Test

Branch 7045b was merged into trunk revision 14899 and archived.

#113 - 01/15/2024 12:42 PM - Greg Shah

See #6667-823 for the needed changes for hand coded Java in the ChUI regression test application.

#114 - 01/23/2024 06:46 AM - Alexandru Lungu

Created 7045c and committed a changed regarding the removal of .next calls inside the BlockManager.forEach body. There are not needed as the block manager was already applying the .next. This problem was not seen in practice; I observed it right now. It could have cause memory leaks or even unexpected behaviors:

```
=== modified file 'src/com/goldencode/p2j/persist/RecordBuffer.java'
--- old/src/com/goldencode/p2j/persist/RecordBuffer.java    2024-01-18 00:21:46 +0000
+++ new/src/com/goldencode/p2j/persist/RecordBuffer.java    2024-01-23 11:43:10 +0000
@@ -1299,6 +1299,7 @@
 **     RAA 20231220          Fixed a case in which ReflectiveOperationException could have been thrown when
 **                           a RecordBuffer was created through a proxy.
 ** 343 OM  20240117          Removed expired comment.
+** 344 AL2 20230123          Removed unwanted .next call inside BlockManager.forEach.
 */

 /*
```

```
@@ -8104,7 +8105,6 @@
              },
              (Body) () ->
              {
-                q.next();
                 try
                 {
                    delete();
@@ -8168,7 +8168,6 @@
           {
              public void body()
              {
-                query.next();
                 try
                 {
                    delete();
```

Please review and let me know if I can merge.

#115 - 01/23/2024 12:32 PM - Greg Shah

Eric: Please review.

#116 - 03/10/2024 07:29 AM - Constantin Asofiei

Alexandru, is 7045c still needed after we refactored the query management to be inside BlockManager? If not, please archive as 'dead'.

Greg: I've created 7045d from trunk rev 15041. In rev 15042, I've added support for Java return from within internal procedures. Please review.

#117 - 03/10/2024 01:41 PM - Alexandru Lungu

Constantin Asofiei wrote:

> Alexandru, is 7045c still needed after we refactored the query management to be inside BlockManager? If not, please archive as 'dead'.

Absolutely; 7045c attempts to remove a "hard" q.next from within the body of a BlockManager.forEach **that uses the query inside already**. The issue is that the next is executed twice (once inside the body and once from withing the BlockManager). I can merge 7045c independently after review **or** let you integrate it in 7045d.

#118 - 03/14/2024 02:46 PM - Constantin Asofiei

Greg, please review 7045d - I'd like to get this into trunk, it doesn't hurt.

#119 - 03/14/2024 02:58 PM - Greg Shah

Code Review Task Branch 7045d Revision 15042

In return_stmts.rules, the type == prog.procedure or on line 93 seems incorrect. It will leave inFunc set to true for procedures.

Otherwise the changes are good.

#120 - 03/14/2024 03:03 PM - Constantin Asofiei

Greg Shah wrote:

> Code Review Task Branch 7045d Revision 15042
>
> In return_stmts.rules, the type == prog.procedure or on line 93 seems incorrect. It will leave inFunc set to true for procedures.

Thanks, rebased and fixed in rev 15061.

I'll do another round of conversion with a small app and I'll put it in my merge queue (probably tomorrow).

#121 - 03/14/2024 03:41 PM - Greg Shah

Sounds good.

#122 - 03/19/2024 11:02 AM - Constantin Asofiei

- Status changed from Test to Merge Pending

Merging this now.

#123 - 03/19/2024 11:04 AM - Constantin Asofiei

- Status changed from Merge Pending to Test

Branch 7045d was merged into trunk as rev. 15068 and archived.

#124 - 03/20/2024 02:54 AM - Constantin Asofiei

- Status changed from Test to Review

Created task branch 7045e from trunk rev 15073. In rev 15074 fixes "RETURN statements from within a CASE statement are not converted to Java return.".

#125 - 03/20/2024 07:34 AM - Greg Shah

- Status changed from Review to Internal Test

Code Review Task Branch 7045e Revision 15074

I'm OK with the change.

#126 - 03/20/2024 12:44 PM - Constantin Asofiei

Greg, can this be merged? It affects a customer's app.

#127 - 03/20/2024 12:45 PM - Greg Shah

If you are confident it is safe, it can be merged now.

#128 - 03/20/2024 01:00 PM - Constantin Asofiei

- Status changed from Internal Test to Test

Branch 7045e was merged into trunk as rev. 15077 and archived.

#129 - 03/21/2024 04:26 AM - Constantin Asofiei

Branch 7045f was created from trunk rev 15077 - in rev 15078 re-fixed the regression in 7045d/e. Merged to trunk rev 15078 and archived.

Files

return_normal_java_before_and_after_20230117.zip	6.88 KB	01/17/2023	Greg Shah
poc_original_3.png	243 KB	12/12/2023	Eduard Soltan
poc_original_snapshot.png	241 KB	12/12/2023	Eduard Soltan
poc_original_1.png	240 KB	12/12/2023	Eduard Soltan
poc_converted_snapshot.png	239 KB	12/12/2023	Eduard Soltan
poc_converted_3.png	241 KB	12/12/2023	Eduard Soltan
poc_converted_2.png	238 KB	12/12/2023	Eduard Soltan