Database - Bug #2955

Update records participating in a scrolling adaptive query.

01/13/2016 04:59 AM - Stanislav Lomany

Status:	WIP	**Start date:**	01/13/2016
Priority:	Normal	**Due date:**	
Assignee:	Stanislav Lomany	**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD		

Description

Related issues:

Related to User Interface - Feature #2564: implement GUI BROWSE widget	**Closed**

History

#1 - 01/13/2016 06:01 AM - Stanislav Lomany

This task was created as the old issue - P2J is sensitive to the content of a buffer participating in the adaptive query - revealed in editing browse.
Testcase 1:

```
  def var i as integer.
  def var j as integer.

  def temp-table tt field f1 as INTEGER
                  index idx1 is unique f1.

  DEF QUERY q FOR tt scrolling.
  def var str as char.

  REPEAT j = 1 TO 10:
   create tt. tt.f1 = j.
  END.
  create tt. tt.f1 = 1000.

  OPEN QUERY q FOR EACH tt.

  get next q.
  get next q.
  get next q.

  tt.f1 = 333.

  get first q.
  message string(tt.f1).
  get next q.
  message string(tt.f1).
  get next q.
  message string(tt.f1).
  get next q.
  message string(tt.f1).
```

4GL output: 1, 2, 333, 4
P2J output: 1, 2, 333, 1000
P2J uses updated value of the record 333 as the placeholder for finding next record.

In 4GL you are free to use the backing buffer to create or update the records. Example:

```
def var i as integer.
```

```
def var j as integer.

def temp-table tt field f1 as INTEGER
                  index idx1 is unique f1.

DEF QUERY q FOR tt scrolling.
def var str as char.

REPEAT j = 1 TO 10:
 create tt. tt.f1 = j.
END.

OPEN QUERY q FOR EACH tt.

get next q.
get next q.
get next q.

create tt. tt.f1 = 99.

get next q.
message string(tt.f1).
get next q.
message string(tt.f1).
get next q.
message string(tt.f1).
```

4GL output: 4, 5, 6
P2J output: 1, 2, 3

There are RecordBuffer.referenceRecord and RAQ.referenceRecord variables, but they doesn't seem applicable for adaptive scrolling query case. I think we should persist the original image of the last record in the cache. What do you think?

#2 - 01/13/2016 01:32 PM - Eric Faulhaber

Stanislav Lomany wrote:

> There are RecordBuffer.referenceRecord

Do you mean RecordBuffer.snapshot?

> ...and RAQ.referenceRecord variables, but they doesn't seem applicable for adaptive scrolling query case. I think we should persist the original image of the last record in the cache. What do you think?

To which cache in which class are you referring, specifically?

#3 - 01/13/2016 03:22 PM - Stanislav Lomany

> Do you mean RecordBuffer.snapshot?

Yes.

> > I think we should persist the original image of the last record in the cache. What do you think?
>
> To which cache in which class are you referring, specifically?

I mean creation of the new field in Cursor which stores a single record which corresponds Cursor.results[results.size - 1].

#4 - 01/13/2016 04:23 PM - Eric Faulhaber

That sounds OK. Your first example suggests that this would have to be a deep copy of the original record (slower) to handle the update case. OTOH, it seems a simple reference to the original record (faster) would suffice for the create/insert case. I'm not sure you have enough information in Cursor to differentiate the cases, however.

Can you help me understand why you concluded that RecordBuffer.snapshot could not be re-purposed for this? It's been a while since I did a careful review of that code, but the original intent was something very similar in terms of query navigation, only w.r.t. deleted/released records instead of

updated/inserted records.

Is your issue specific to adaptive, scrolling queries?

#5 - 01/14/2016 07:55 PM - Stanislav Lomany

> That sounds OK. Your first example suggests that this would have to be a deep copy of the original record (slower) to handle the update case. OTOH, it seems a simple reference to the original record (faster) would suffice for the create/insert case. I'm not sure you have enough information in Cursor to differentiate the cases, however.

Maybe we can monitor the target record for updates and copy it only if an update has been performed? Also, we can deep copy only the fields which are used to walk along the index.

> Can you help me understand why you concluded that RecordBuffer.snapshot could not be re-purposed for this? It's been a while since I did a careful review of that code, but the original intent was something very similar in terms of query navigation, only w.r.t. deleted/released records instead of updated/inserted records.

In 4GL we are free to use a buffer in multiple queries and CRUD operations without affecting walk order for adaptive scrolling queries, so saving the snapshot in RecordBuffer is not good because it will be affected by these operations.

> Is your issue specific to adaptive, scrolling queries?

Yes.

#6 - 01/19/2016 08:00 AM - Eric Faulhaber

Please go ahead with your proposed approach.

#7 - 02/02/2016 08:02 AM - Stanislav Lomany

- Status changed from New to WIP

There are more issues with compound adaptive queries than I've expected. When a backing buffer is updated, the following records returned by the

query differs in 4GL and P2J. This applies both to scrolling and non-scrolling queries. To trigger this kind of issues in a real application you should edit the last row in a browse (following records shouldn't be loaded yet, doesn't apply to the initial browse view).
Testcase:

```
def var i as integer.

def temp-table tt field f1 as INTEGER
                  field f2 as INTEGER
                  index idx1 is unique f1 f2.

def temp-table tt2 field g1 as INTEGER
                   index idx2 is unique g1.

def temp-table tt3 field k1 as INTEGER
                   index idx3 is unique k1.

DEF QUERY q FOR tt, tt2, tt3.

create tt. tt.f1 = 2.
create tt. tt.f1 = 4.
create tt. tt.f1 = 6.

create tt2. tt2.g1 = 2.
create tt2. tt2.g1 = 4.
create tt2. tt2.g1 = 6.

create tt3. tt3.k1 = 2.
create tt3. tt3.k1 = 4.
create tt3. tt3.k1 = 6.

OPEN QUERY q FOR EACH tt, each tt2 where tt.f1 <> tt2.g1, each tt3 where tt.f1 <> tt3.k1.

get next q.
get next q.
get next q.
get next q.
get next q.

tt.f1 = 10.
tt2.g1 = 10.
tt3.k1 = 10.

get next q.
repeat:
   message string(tt.f1) + " " + string(tt2.g1) + " " + string(tt3.k1).
   get next q.
   if not avail(tt) then leave.
end.
```

In this piece of data

```
4 2 2
4 2 6
4 6 2
4 6 6
```

we update the first row to 10 10 10. Next rows in 4GL:

```
10 10 6
10 10 10
10 6 4
10 6 6
10 10 4
10 10 6
```

Next rows in P2J:

```
10 10 4
10 10 6
10 4 4
10 4 6
10 6 4
10 6 6
```

#8 - 02/02/2016 08:51 AM - Eric Faulhaber

Stanislav Lomany wrote:

> There are more issues with compound adaptive queries than I've expected.

When you write, "compound adaptive queries", are you referring to the optimization I added recently to back CompoundQuery with a server-side join? If so, I think we can just disable this optimization when the query is used in a browse. Would that simplify this issue?

#9 - 02/02/2016 09:03 AM - Stanislav Lomany

Can you please tell me how to turn it off? But the issues isn't about optimization, it presents in the bzr versions before it was introduced.

#10 - 02/02/2016 10:15 AM - Eric Faulhaber

To disable the server-side join optimization, change the implementation of CompoundQuery.canOptimize.

#11 - 02/24/2016 10:44 AM - Stanislav Lomany

Consider the current record was updated at some level of an adaptive compound query. Then a query at more granular level uses the snapshot of this record (value before update) as the selection criteria *until* the next loop starts at this level.
I'm working on that.

#12 - 03/02/2016 10:36 AM - Stanislav Lomany

- Assignee set to Stanislav Lomany

Guys, please consider the design of the solution for this issue (see previous note):
1. Register compound query as a record change listener. If the current record in a backing buffer was updated - update query state accordingly.
2. Create ParameterFilter interface with the single method filterParameters.
3. Add setParameterFilter(ParameterFilter) method to P2JQuery interface.
4. Create anonymous classes in CompoundQuery and assign them as parameter filters for its components (preselect or adaptive queries).
5. When a component's query is executed to get the result set at specific point, it uses assigend parameter filters to filter substitution parameters and use original field references as is or use snapshot values (RecordBuffer.snapshot) instead. The latter used if a) the substitutuion parameter belongs to a backing buffer of an outer component AND b) if the state in the parent compound query tells us to use snapshot values.

What do you think?

#13 - 03/02/2016 11:22 AM - Eric Faulhaber

Stanislav Lomany wrote:

> Consider the current record was updated at some level of an adaptive compound query. Then a query at more granular level uses the snapshot of this record (value before update) as the selection criteria *until* the next loop starts at this level.

Sorry, are you describing an incorrect behavior above, or are you describing what the correct behavior should be?

#14 - 03/02/2016 02:29 PM - Stanislav Lomany

That is how it works in 4GL.

#15 - 03/02/2016 10:49 PM - Eric Faulhaber

Stanislav Lomany wrote:

> Guys, please consider the design of the solution for this issue (see previous note):
> 1. Register compound query as a record change listener. If the current record in a backing buffer was updated - update query state accordingly.
> 2. Create ParameterFilter interface with the single method filterParameters.
> 3. Add setParameterFilter(ParameterFilter) method to P2JQuery interface.
> 4. Create anonymous classes in CompoundQuery and assign them as parameter filters for its components (preselect or adaptive queries).
> 5. When a component's query is executed to get the result set at specific point, it uses assigend parameter filters to filter substitution parameters and use original field references as is or use snapshot values (RecordBuffer.snapshot) instead. The latter used if a) the substitutuion parameter belongs to a backing buffer of an outer component AND b) if the state in the parent compound query tells us to use snapshot values.
>
> What do you think?

If I understood correctly, the ParameterFilter part of the proposed solution is only about reversing the optimization done with the FieldReference parameters when optimizing the compound query to a server-side query, right? Does this complexity go away if we just disallow server-side optimization for compound queries which back an editable browse? Or am I missing why this would be necessary in the unoptimized compound query case?

#16 - 03/03/2016 12:53 PM - Stanislav Lomany

This issue applies no scrolling/non-scrolling compound queries with no regards to browse (however using an editable browse is the easiest way to expose it). One more testcase:

```
def var i as integer.

def temp-table tt field f1 as INTEGER
                 index idx1 is unique f1.

def temp-table tt2 field g1 as INTEGER
                  field g2 as INTEGER
                  index idx2 is unique g1 g2.

DEF QUERY q FOR tt, tt2.

create tt. tt.f1 = 1.

create tt2. tt2.g1 = 1. tt2.g2 = 1.
create tt2. tt2.g1 = 1. tt2.g2 = 2.
create tt2. tt2.g1 = 1. tt2.g2 = 3.

create tt2. tt2.g1 = 2. tt2.g2 = 1.
create tt2. tt2.g1 = 2. tt2.g2 = 2.
create tt2. tt2.g1 = 2. tt2.g2 = 3.

OPEN QUERY q FOR EACH tt, each tt2 where tt.f1 = tt2.g1.

get first q.

tt.f1 = 2.
tt2.g1 = 4.

repeat:
   message string(tt.f1) + " " + string(tt2.g1) + " " + string(tt2.g2).
   get next q.
   if not avail(tt) then leave.
end.
```

P2J output:

```
2 4 1
2 2 1
2 2 2
2 2 3
2 2 1
2 2 2
2 2 3
```

4GL output:

```
2 4 1
2 1 2
2 1 3
2 2 1
2 2 2
2 2 3
```

I'm not sure how this testcase will work if optimization is on because it abends. Also I don't know how to unite optimization and this kind of behavior.

#17 - 03/03/2016 02:27 PM - Eric Faulhaber

Stanislav Lomany wrote:

> This issue applies no scrolling/non-scrolling compound queries with no regards to browse (however using an editable browse is the easiest way to expose it).

OK, I was just trying to understand the scope of the problem. If you still think your proposed implementation will solve the problem, I'm ok with it.

> I'm not sure how this testcase will work if optimization is on because it abends.

How? What's the error?

> Also I don't know how to unite optimization and this kind of behavior.

I'm not sure what you mean, but the idea behind optimization was to push as much processing to the database server as possible by using a join instead of the nested looping that CQ does by default. The intention was to revert back to dynamic mode in the event of an update which affects an index which any CQ component relies upon for its results, as if the query had been operating by the dynamic approach all along. I guess you already know all this, but I'm not sure what you're asking, specifically.

#18 - 03/03/2016 02:56 PM - Stanislav Lomany

> How? What's the error?

```
Caused by: java.lang.UnsupportedOperationException
        at com.goldencode.p2j.persist.CompoundQuery.getBreakValue(CompoundQuery.java:1502)
        at com.goldencode.p2j.persist.AdaptiveQuery$DynamicResults.postFetch(AdaptiveQuery.java:3932)
        at com.goldencode.p2j.persist.AdaptiveQuery$DynamicResults.next(AdaptiveQuery.java:3723)
        at com.goldencode.p2j.persist.ResultsAdapter.next(ResultsAdapter.java:119)
        at com.goldencode.p2j.persist.AdaptiveQuery.next(AdaptiveQuery.java:1813)
        at com.goldencode.p2j.persist.CompoundQuery.processComponent(CompoundQuery.java:2453)
        at com.goldencode.p2j.persist.CompoundQuery.retrieveImpl(CompoundQuery.java:2125)
        at com.goldencode.p2j.persist.CompoundQuery.retrieve(CompoundQuery.java:1674)
        at com.goldencode.p2j.persist.CompoundQuery.retrieve(CompoundQuery.java:1561)
        at com.goldencode.p2j.persist.CompoundQuery.next(CompoundQuery.java:952)
        at com.goldencode.p2j.persist.CompoundQuery.next(CompoundQuery.java:854)
```

```
        at com.goldencode.p2j.persist.QueryWrapper.next(QueryWrapper.java:1956)
```

If I understand correctly, in the optimized case of this testcase tt.f1 = tt2.g1 is a server-side join, so the database value is used for tt.f1 and my concern is that I have no control over it: I cannot specify if the actual or snapshot value should be used for specific case.

#19 - 03/03/2016 03:14 PM - Eric Faulhaber

Stanislav Lomany wrote:

> How? What's the error?

```
Caused by: java.lang.UnsupportedOperationException
        at com.goldencode.p2j.persist.CompoundQuery.getBreakValue(CompoundQuery.java:1502)
        at com.goldencode.p2j.persist.AdaptiveQuery$DynamicResults.postFetch(AdaptiveQuery.java:3932)
        at com.goldencode.p2j.persist.AdaptiveQuery$DynamicResults.next(AdaptiveQuery.java:3723)
        at com.goldencode.p2j.persist.ResultsAdapter.next(ResultsAdapter.java:119)
        ...
```

Ah, that's actually one that occurs in unit testing and has been reported by the customer. Looks like an oversight on my part when implementing the optimization. When the code that throws this exception was written long before, it was supposed to be a place that was never invoked. That changed with the optimization. This has to be fixed, but not necessarily as part of this task.

> If I understand correctly, in the optimized case of this testcase tt.f1 = tt2.g1 is a server-side join, so the database value is used for tt.f1 and my concern is that I have no control over it: I cannot specify if the actual or snapshot value should be used for specific case.

Yes, in the optimized case, we've refactored the query so that tt.f1 is no longer a FieldReference object, but is handled as part of the server-side join. However, the optimization should be undone by the updates, and you should have access to the original compound query and its components. The buffer backing tt1 should have been updated regardless of whether we were in optimized or original mode. Am I missing something?

#20 - 03/16/2016 02:35 PM - Stanislav Lomany

Eric, could you please provide a testcase with an *optimized* adaptive compoud query that works? I coundn't find one in bzr or think it out.

#21 - 03/16/2016 02:59 PM - Eric Faulhaber

I think the one I used was compound_query_test.p, but it looks like I had not committed it previously (I've done so now). That contains several nested FOR statements which IIRC will optimize with various degrees of success. Unfortunately, I don't remember which one(s) optimize. So, try this: convert that test case and run with a com.goldencode.p2j.persist.CompoundQuery logger set to FINE. This will log detailed information about the optimization process. Look for the message +++ passed all validation, then scroll up to see with which FOR loop the log message is associated.

#22 - 03/17/2016 04:52 PM - Stanislav Lomany

FYI, I've fixed some cases revealed by this testcases but eventually I encountered getBreakValue issue for every case. I'll return to optimized queries later if it will be necessary.

#23 - 04/08/2016 08:51 PM - Stanislav Lomany

The update was implemented in 2564f and committed into the trunk as bzr revision 10999.
We may want to re-visit this task after optimized queries will be fixed.

#24 - 04/09/2016 03:35 AM - Greg Shah

- Project changed from User Interface to Database

#25 - 04/09/2016 11:09 AM - Eric Faulhaber

Stanislav Lomany wrote:

> We may want to re-visit this task after optimized queries will be fixed.

The UnsupportedOperationException on CompoundQuery.getBreakValue is fixed in task branch 2475a, which I am regression testing now.

#26 - 04/25/2016 02:16 PM - Eric Faulhaber

Eric Faulhaber wrote:

> Stanislav Lomany wrote:
>
> > We may want to re-visit this task after optimized queries will be fixed.
>
> The UnsupportedOperationException on CompoundQuery.getBreakValue is fixed in task branch 2475a, which I am regression testing now.

This issue is now fixed in the trunk (rev. 11009).