Database - Feature #8067

investigate adding automatically adding a FIELDS clause for simple record blocks

11/21/2023 09:28 AM - Constantin Asofiei

Status:	WIP	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Eduard Soltan	**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**vendor_id:**	GCD

Description

Related issues:

Related to Database - Feature #6721: calculate field lists at conversion time...	**New**

History

#1 - 11/21/2023 09:31 AM - Constantin Asofiei

In a customer application, we noticed patterns like for each tt1: do something with a few fields. end. where although we retrieve the entire record (persistent or temp-table), only some fields are being actually read.

No FIELDS clause exists in these cases, and also for temp-tables the FIELDS clause is ignored completely in 4GL. The idea here is this: do some static analysis and, if some conditions are met (i.e. no calls are performed via RUN/etc, no dynamic access), then force a 'fieldsInternal' clause to retrieve only this partial record; this would entail allowing support for FIELDS with temp-tables, and also to discuss/analyze implications on before-table or triggers if an incomplete record is being used (Alexandru, we were wondering about schema triggers, FIELDS is used only if NO-LOCK, otherwise is ignored in 4GL). This would help in terms that we can retrieve via JDBC only the actually read fields, and not the entire record (even with lazy hydration JDBC still retrieves the entire record).

This is not solved automatically via lazy hydration, as JDBC still retrieves the entire SELECT clause from the database, is just FWD delays deserializing the fields only when they are truly needed. FIELDS works directly at the SELECT clause, which would result in the JDBC driver actually retrieving less fields from the database.

#2 - 11/21/2023 09:47 AM - Greg Shah

We can certainly use TRPL to analyze field usage and generate an implicit FIELDS clause if certain constraints are met.

Is it worth mocking up the difference to see the effect? This could be tested by 2 versions of the same 4GL code, one with a manually created FIELDS and one without. Unfortunately, this will only work with smaller testcases and not a larger application. I would expect this to show more benefits for "wide" tables.

#3 - 11/21/2023 09:51 AM - Constantin Asofiei

Greg Shah wrote:

> We can certainly use TRPL to analyze field usage and generate an implicit FIELDS clause if certain constraints are met.

A report to analyze the .ast files and see how many blocks the implicit FIELDS can be applied should be easy to write. The runtime part is a little more complicated, especially for temp-tables, as an incomplete record will affect the before-table, so at least that needs to be considered.

#4 - 11/21/2023 12:26 PM - Greg Shah

Let's move ahead with this. Note that we should do this analysis **after** record scoping is finished in annotations. One important contraint to include is that the buffer must be scoped to the FOR loop. If the scope is larger than that, then the analysis would have to be broader as well.

#5 - 11/21/2023 12:29 PM - Greg Shah

- Related to Feature #6721: calculate field lists at conversion time where possible added

#6 - 11/21/2023 12:56 PM - Constantin Asofiei

Greg Shah wrote:

> One important contraint to include is that the buffer must be scoped to the FOR loop. If the scope is larger than that, then the analysis would have to be broader as well.

There are some cases here:

- if the FOR block executes completely, the buffer will remain with no record, regardless of scope
- if the FOR block terminates early, and the buffer is scoped to some outer block, then the buffer will remain populated with that record. And this is the part that is important - we can't 'leak' these kind of incomplete records to the 'outside world'.

In Openedge, a FIELDS incomplete record from a FOR block can be referenced outside the FOR block. For our 'internal fields' case, once the block scope exits, if the buffer still references an incomplete 'internal fields' record, we can force to load it fully. IMO, this can be considered an edge case.

#7 - 11/21/2023 01:09 PM - Greg Shah

That is why I was suggesting bypassing that case.

#8 - 11/21/2023 01:16 PM - Constantin Asofiei

Greg Shah wrote:

> That is why I was suggesting bypassing that case.

I agree, but this would exclude all temp-tables, as currently FWD scopes them to the external program (or the class), always.

#9 - 11/21/2023 02:07 PM - Greg Shah

Good point.

#10 - 11/21/2023 02:22 PM - Constantin Asofiei

A crude report on the .ast of a large POC app results in ~13% (18k from 141k) of the FOR EACH blocks which can be improved with 'internal fields'. Although this still need to be refined so that:

- property calls are not allowed (but I think we can at least process the property to see if this is an implicit getter/setter)
- enum calls are allowed
- skeleton calls are allowed, as these can't use the buffer directly (except some cases I think like JSON write, but IIRC these work only with full temp-table)
- bffer-copy statements, buffer record write json/xml, need the entire record (there may be others)
- ::, buffer-value and other cases which use the buffer directly.

IMO, this looks promising.

#11 - 11/23/2023 05:20 AM - Alexandru Lungu

Constantin, what kind of queries did you consider. FOR EACH loops with BREAK BY shouldn't be counted (or maybe can be, but add to the FIELDS the fields from the BY so we can do the server-side sorting). Also, mind the FOR EACH blocks with client-where.

#12 - 11/23/2023 06:03 AM - Constantin Asofiei

I have not went further with this. The report I used is this:

```
<cfg>

   <!-- register worker objects -->
   <worker class="com.goldencode.p2j.uast.ProgressPatternWorker"      namespace="prog" />

   <!-- expression libraries -->
   <include name="common-progress" />

   <!-- main processing (once per AST) -->
   <rule-set input="tree" >

      <descent-rules>
         <variable name="bla" type="java.lang.Long" init="#(long) 12345"/>
         <variable name="foo" type="java.lang.String" init="'0'"/>
         <variable name="match" type="java.lang.Boolean" init="false"/>
         <rule>match = false</rule>

         <rule>this.type == prog.block and parent.type == prog.inner_block and this.prevSibling.type == prog.k
w_for and this.prevSibling.downPath(prog.kw_each)
            <action>addScope("for_fields")</action>
            <action>addDictionaryBoolean("for_fields", "fields", true)</action>
         </rule>
      </descent-rules>

      <walk-rules>
         <rule>
            dictionaryMaxDepth("for_fields") &gt; 1 and
            ((type == prog.kw_run and parent.type == prog.statement)       or
             evalLib("call_to_user_defined_function", this)                or
             evalLib("oo_method_calls")                                    or
             evalLib("type_pair", this, prog.func_class, prog.kw_dyn_new)  or
             evalLib("type_pair", this, prog.func_class, prog.kw_new))

            <!-- TODO: property calls are not allowed -->
            <!-- TODO: enum calls are allowed -->
            <!-- TODO: skeleton calls can are allowed, as these can't use the buffer directly -->
            <!-- TODO: copy-lob statements need the entire record -->

            <!-- only FOR EACH blocks, as FOR FIRST/LAST leaves behind a record -->
            <action>printfln("ignored %s %s %s %s", dictionaryMaxDepth("for_fields"), this.filename, this.id,
this.line)</action>
```

```
            <action>addDictionaryBoolean("for_fields", "fields", false)</action>
         </rule>
      </walk-rules>

      <ascent-rules>
         <rule>this.type == prog.block and parent.type == prog.inner_block and this.prevSibling.type == prog.k
w_for and this.prevSibling.downPath(prog.kw_each)
            <rule>lookupDictionaryBoolean("for_fields", "fields")
               <action>printfln("used %s %s:%s", this.filename, this.prevSibling.line, this.prevSibling.column
)</action>
            </rule>
            <action>deleteScope("for_fields")</action>
         </rule>
      </ascent-rules>
   </rule-set>

</cfg>
where:
* @ignored@ means the FOR EACH block can't have the implicit FIELDS added
* @used@ means we can emit the implicit FIELDS

The solution needs further work, more refining of the blocks being used and also to collect the actual field l
ist.
```

#13 - 12/04/2023 10:25 AM - Alexandru Lungu

- Status changed from New to WIP

- Assignee set to Eduard Soltan

Constantin, can you provide some guidance on how to run your rule-set against a large customer application without actually integrating them in our rules and do a full conversion? This is a report and can be visualized using our report engine. Is there maybe an ant script we have to run only this rule over the .ast files we have at command-line?

Eduard, please update yourself with the topic here. We are really interested in the potential of a solution in which we fetch less fields from the database by synthetically generating FIELDS clauses. This is 100% conversion work from my POV, as we already have "incomplete" record support on the run-time part that will handle the FIELDS clause.

#14 - 12/04/2023 10:47 AM - Constantin Asofiei

This command assumes the .xml file is saved as rules/reports/fields.xml and included in a p2j.jar build:

```
java -server -Xmx32g -classpath p2j/build/lib/p2j.jar com.goldencode.p2j.pattern.PatternEngine reports/fields
```

```
./abl "*.ast"  2>&1 | tee "fields.log"
```

This is not a FWD analytics report. You need to interpret the fields.log file by hand.

#15 - 12/15/2023 07:43 AM - Eduard Soltan

Committed on 8967a, revision 14872.

Added an annotation internalfields with fields used in a FOR EACH block to kw_buffer/kw_table nodes that have as parent a kw_for node and at least a kw_each sibilling.
On "Code Core Conversion" phase, in Init Block of the forEach method, for every buffer of the forEach query added setInclude method with the fields from internalfields annotation.

This task will probably also require runtime change.
From what I seen, at PreselectQuery initial query is performed with just recid in select cluse. After that in coreFetech, the rest of the fields are loaded in the buffer using Persistance.load.

Call to Persistance.load is made with the partialFields set on null, so all the fields are retrieve. So all fields are reterive even if included data member is not empty.

#16 - 12/21/2023 11:13 AM - Eric Faulhaber

Please note that an explicit FIELDS clause in 4GL code will raise an error if a field is accessed in downstream code which was not included in the FIELDS clause. If FWD is automagically adding a FIELDS clause, we have to be sure that either:

- the static analysis was spot-on or sufficiently conservative, such that we don't get into a situation where a downstream field access raises an error condition that normally would not occur in OE; or
- we account in the runtime for the difference between an explicit/original FIELDS clause and one that FWD adds, to avoid this situation.

#17 - 12/21/2023 11:23 AM - Alexandru Lungu

I fully agree with Eric and mark this as top focus in the implementation. However, I am not 100% sure this approach will lead to improvement. Having such FIELDS constructs will determine more than 1 SQL to load a DMO. Thus, we may end up with a fresh short SQL to be parsed instead of using a long prepared SQL. I think we should focus on implementing this at the level at which we can test performance. Is it really worth it to generate new shorter fresh SQLs just to retrieve less data from the database? Or should we stick with the current "load everything" just to avoid reparsing.

Also, the session cache will have a worse ratio, as there will be way more incomplete records that can't be reused. I've seen a problem before here and I think we should address it at some point - don't refetch an incomplete DMO if the provided rowStructure doesn't require it. Currently, we rehydrate an incomplete DMO each time we retrieve it from the cache (AFAIK).