Database - Feature #6459

query features

05/27/2022 12:59 PM - Eric Faulhaber

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Igor Skornyakov	**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**vendor_id:**	GCD

Description

Related issues:

Related to Database - Bug #2095: FIELDS/EXCEPT record phrase options not honored	**Closed**
Related to Database - Feature #2137: runtime support for FIELDS/EXCEPT record...	**New**

History

#1 - 05/27/2022 02:45 PM - Eric Faulhaber

Implement and improve the following:

- QUERY handle
  - FIRST-OF, LAST-OF methods (gap marked none/none) - AFAIK we support the core features, but we probably don't have the method forms implemented
  - CACHE attribute (gap marked runtime stubs) - This represents how many records of a NO-LOCK query are held in memory.
- static query options
  - QUERY-TUNING phrase is not supported. Adding support depends on the backing database, and the tuning choices may not translate well between the Progress database and the new RDBMS. This needs further exploration.
  - INDEXED-REPOSITION (see #5673)
  - MAX-ROWS
- record phrase options
  - FIELDS currently is a no-op which is dropped at conversion, but we should implement
  - TABLE-SCAN is not supported. Even if we have a backing database which supports this as a query plan hint, it is unknown whether the optimization will translate from the Progress DB to the new RDBMS.
  - OUTER-JOIN
  - NO-PREFETCH (we probably can ignore this, but we need to update our gap marking; see notes below)

Some notes on the QUERY:CACHE attribute:

I'm not sure why it is limited to NO-LOCK (at least, that's what is documented). We should try to confirm that this is really the case.

I'd like to avoid an implementation which changes the row limits we set when composing FQL/SQL, because doing this will necessitate implementing messy paging logic and probably a more expensive implementation (using multiple queries instead of letting the JDBC driver do lower level result set paging). I'd much rather let the JDBC driver handle this on our behalf. So I'm hoping we can implement this using JDBC fetch size.

We usually default JDBC fetch size to a pretty high value for JDBC connections (usually 1024, set in the directory), but this also can be set at the statement level, which I understand overrides the connection setting.

In theory, the higher the fetch size, the fewer round trips are needed to process large result sets, but this efficiency comes at the expense of JVM heap in the FWD server. I say, "in theory", because JDBC fetch size is only a *hint* to the JDBC driver. We need to confirm that the JDBC driver we are using for any given database honors it, else this setting will be ignored. There may be other requirements on the query options in use, such as using a forward-scrolling cursor, which may conflict with result set navigation in certain common cases.

Some notes on the QUERY-TUNING and TABLE-SCAN options:

The QUERY-TUNING phrase is documented to support only DataServer queries, not queries against the Progress database. Nevertheless, these currently are not supported, due to the impedance mismatch inherent in queries between the 4GL and FWD, in terms of the SQL ultimately executed, even if the back-end RDBMS is the same.

To the degree the original 4GL query is designed for use with a DataServer of the same database type as we have configured with FWD at runtime, we *might* find something useful to implement here. That being said, the SQL we ultimately send to the backing database from FWD for a given 4GL query is pretty much guaranteed to be significantly different than the original SQL. Consider that we convert each such query to a Java query object in FWD, in which the statement is expressed in FQL. A join query will almost certainly be very different, as it may be broken into pieces (CompoundQuery), or even if converted to PreselectQuery, is likely to be assembled differently (first as FQL and ultimately as SQL).

Beyond that, some databases (for instance, PostgreSQL) do not enable per-query tuning hints at all. Other RDBMSs do accept hints, but the same tuning isn't likely to apply across different database implementations, even if the old and new SQL were identical (which it won't be, see above). A SQL RDBMS will make its own query planning decisions via its own algorithms, based on available indices, data statistics, query structure, cost-based configuration, available memory, etc. So matching the DataServer type with the FWD database server type would be a minimum requirement. I am lumping the TABLE-SCAN option with other query tuning settings, since the decision whether or not to do a table scan vs. index scan is just another part of a query plan.

Nevertheless, these options require some exploration, to see if there seems to be any useful translation we can do with this tuning information.

Some notes on the OUTER-JOIN option:

This option is partially supported, in that outer join semantics are honored when OUTER-JOIN is specified, but this option currently will force conversion to CompoundQuery. Even a PRESELECT query (4GL) will not convert to PreselectQuery (FWD) if the OUTER-JOIN option is used. This is for historical reasons, and the limitation which required this no longer exists in FWD, but the conversion of PRESELECT ... OUTER-JOIN queries to CompoundQuery has not yet been fixed to reflect this.

Note that currently, simply using FOR EACH/FIRST/LAST with joined tables still will convert to CompoundQuery. The FOR form is used much more often than the PRESELECT form in most 4GL code we've seen, so eliminating the limitation with PRESELECT will not buy us much. The performance of CompoundQuery is a separate issue, not in the scope of this task.

Some notes on the NO-PREFETCH option:

When used with FIND and CAN-FIND, this matches our default behavior already, so we can (and currently do) ignore this option. This option conflicts with our implementation of AdaptiveQuery and PreselectQuery, but it also does not seem to make a lot of sense to use this option in the context of FOR EACH or PRESELECT type queries, so it probably is OK to continue to ignore this. However, we need to update gap marking to reflect that this feature effectively is implemented (always) for [CAN-]FIND and won't be implemented for the other types. Maybe we just mark it full/full?

#2 - 07/28/2022 02:01 PM - Eric Faulhaber

- Related to Bug #2095: FIELDS/EXCEPT record phrase options not honored added

#3 - 07/28/2022 02:01 PM - Eric Faulhaber

- Related to Feature #2137: runtime support for FIELDS/EXCEPT record phrase options added

#4 - 11/16/2022 01:01 PM - Eric Faulhaber

- Assignee set to Igor Skornyakov

Igor, please focus only on the first bullet point (i.e., QUERY handle) items.

#5 - 11/18/2022 08:37 AM - Igor Skornyakov

As far as I can see from my test, FIRST-OF, LAST-OF methods are fully supported.
However, I see the following issue.
The loop in the following:

```
DEFINE TEMP-TABLE tt NO-UNDO
    FIELD fint AS INT
    FIELD fchar1 AS CHAR
    FIELD fchar2 AS CHAR
    FIELD fchar AS CHAR
    INDEX idx AS PRIMARY fint
.

// populate tt

DEFINE QUERY qtt FOR tt SCROLLING.

QUERY qtt:QUERY-PREPARE("FOR EACH tt BREAK BY tt.fchar1 BY tt.fchar2").
QUERY qtt:QUERY-OPEN.

REPEAT:
  GET NEXT qtt.
  IF NOT AVAILABLE tt THEN LEAVE.
END.
```

never ends (AVAILABLE tt) is always true.

Should I try to fix it now?
Thank you.

#6 - 11/18/2022 09:17 AM - Greg Shah

> As far as I can see from my test, FIRST-OF, LAST-OF methods are fully supported.

At a minimum, the gap marking needs to be fixed since it is set to none/none.

Eric will have to answer on the other bug.

#7 - 11/18/2022 09:42 AM - Igor Skornyakov

Greg Shah wrote:

> As far as I can see from my test, FIRST-OF, LAST-OF methods are fully supported.

> At a minimum, the gap marking needs to be fixed since it is set to none/none.
>
> Eric will have to answer on the other bug.

In 6129b both FIRST-OF and LAST-OF are set to rw.cvt_lvl_full | rw.rt_lvl_full

#8 - 11/18/2022 10:05 AM - Eric Faulhaber

Igor Skornyakov wrote:

> In 6129b both FIRST-OF and LAST-OF are set to rw.cvt_lvl_full | rw.rt_lvl_full

The function form is marked full. The method form (e.g., see attrs.put(prog.kw_first_of, rw.cvt_lvl_none | rw.rt_lvl_none)) is marked none. Are your tests using the function form or the method form?

#9 - 11/18/2022 10:20 AM - Greg Shah

The convert/methods_attributes.rules has support for both methods. Subclasses of P2JQuery like Persorter have apparently full implementations of isFirstOfGroup()/isLastOfGroup(). I agree it should be tested, but on the surface a 5 minute check shows some support be there.

#10 - 11/18/2022 10:21 AM - Igor Skornyakov

Eric Faulhaber wrote:

> Igor Skornyakov wrote:
>
> > In 6129b both FIRST-OF and LAST-OF are set to rw.cvt_lvl_full | rw.rt_lvl_full
>
> The function form is marked full. The method form (e.g., see attrs.put(prog.kw_first_of, rw.cvt_lvl_none | rw.rt_lvl_none)) is marked none. Are your tests using the function form or the method form?

Sorry for my ignorance, but what is the difference?
I was testing the query object handle methods described e.g. here:
https://docs.progress.com/en-US/bundle/openedge-abl-reference-117/page/FIRST-OF-method.html

#11 - 11/18/2022 11:42 AM - Greg Shah

There are 2 forms of these: built-in function and handle-based method.

Built-In Function form:

```
for each whatever BREAK BY whatever.some-field:
   if FIRST-OF(whatever.some-field) then
...
end.
```

Method form:

```
def query q for whatever scrolling.

query q:query-prepare("for each whatever BREAK BY whatever.some-field").
query q:query-open().

...
   get next q.

   if query q:FIRST-OF(0) then
...
```

The handle-based method forms are what need to be supported and are at least partially supported already.

#12 - 11/18/2022 12:13 PM - Igor Skornyakov

Greg Shah wrote:

> There are 2 forms of these: built-in function and handle-based method.
>
> Built-In Function form:

> [...]
>
> Method form:
>
> [...]
>
> The handle-based method forms are what need to be supported and are at least partially supported already.

I see, thank you. According to my test handle-based method forms are already fully supported.

#13 - 11/22/2022 08:41 AM - Igor Skornyakov

- File available.diff added

According to my tests, both forms of FIRST-OF/LAST-OF are fully supported.

I've also fixed the problem with available described in #6459-5 (see attachment).
Please review.
Thank you.

#14 - 11/24/2022 09:42 AM - Igor Skornyakov

I do not see how we can reliably figure out how the CACHE attribute value affects the performance. Such things usually depend on many factors - environment, application load, or even application logic. Moreover, the same value can have a dramatically different effect with 4GL and FWD.
With this in mind, I suggest ignoring this setting in FWD.
However, maybe it makes sense to allow adjustment of the cache size for a particular query on a try-and-see basis by providing e.g. JMX bean for this.
The only thing is how to specify which bean is attached to which query. At this moment I cannot suggest a good solution (maybe a conversion hint?).
What do you think?
Thank you.

#15 - 11/25/2022 05:24 AM - Constantin Asofiei

Igor Skornyakov wrote:

> According to my tests, both forms of FIRST-OF/LAST-OF are fully supported.
>
> I've also fixed the problem with available described in @6459-5 (see attachment).
> Please review.
> Thank you.

Releasing the record in throwOFfend I don't think is correct; see this from PreselectQuery.fetch.

```
         // under certain circumstances, we must bypass releasing the current record(s) (if any)
```

```
        // in the buffer(s), so we throw the off-end exception directly instead of setting the
        // current buffer record to null
        if (!available   &&
            rowCount < 0 &&
            !errorIfNull &&
            _isOffEnd()  &&
            preserveBuffersOnEmptyResults())
        {
           getRecordBuffers()[0].throwOffEnd();
        }
```

What's the stacktrace where #6459-5 calls throwOffend()?

#16 - 11/25/2022 05:59 AM - Igor Skornyakov

Constantin Asofiei wrote:

> What's the stacktrace where #6459-5 calls throwOffend()?

Here it is:

```
Daemon Thread [Conversation [00000007:bogus]] (Suspended (breakpoint at line 9198 in RecordBuffer))
    TemporaryBuffer(RecordBuffer).throwOffEnd() line: 9198
    PresortQuery(PreselectQuery).fetch(boolean, LockType, boolean) line: 5655
    PresortQuery(PreselectQuery).next(LockType) line: 2612
    PresortQuery(PreselectQuery).next() line: 2568
    FirstLastOf.lambda$null$1(PresortQuery, FieldReference, FieldReference) line: 93
```

FirstLastOf is my test program.

#17 - 11/25/2022 11:08 AM - Eric Faulhaber

Igor, I'm looking into the history of this part of the code. I don't think it's as simple as releasing the record unconditionally. There was some reason I found (ca. rev 10959.1.6) to not release the record in at least one BREAK BY case, though your test would suggest the buffer needs to be NOT available. I'm trying to figure out what the differentiating factor was...

#18 - 11/25/2022 01:35 PM - Igor Skornyakov

Eric Faulhaber wrote:

> Igor, I'm looking into the history of this part of the code. I don't think it's as simple as releasing the record unconditionally. There was some reason I found (ca. rev 10959.1.6) to not release the record in at least one BREAK BY case, though your test would suggest the buffer needs to be NOT available. I'm trying to figure out what the differentiating factor was...

Eric,
I've got the impression that the problem is not related to the number of BREAK BY clauses. It is just a specific of the PresortQuery (or maybe even more generic).
I recall another problem with available when we report it as true even if next() results in the exception thrown by a UDF (we have a separate task for it but I cannot recall its number at this moment).
However, you understand the logic much better of course.

#19 - 11/25/2022 05:40 PM - Greg Shah

Eric Faulhaber wrote:

> Igor, I'm looking into the history of this part of the code. I don't think it's as simple as releasing the record unconditionally. There was some reason I found (ca. rev 10959.1.6) to not release the record in at least one BREAK BY case, though your test would suggest the buffer needs to be NOT available. I'm trying to figure out what the differentiating factor was...

It was related to #2655.

Files

available.diff	431 Bytes	11/22/2022	Igor Skornyakov