Database - Feature #2137

runtime support for FIELDS/EXCEPT record phrase options

04/26/2013 09:28 AM - Eric Faulhaber

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	80%
Category:		**Estimated time:**	80.00 hours
Target version:			
billable:	No	**version:**	
vendor_id:	GCD		

Description

Related issues:

Related to Database - Bug #2095: FIELDS/EXCEPT record phrase options not honored	**Closed**
Related to Database - Feature #6459: query features	**New**
Related to Database - Feature #6721: calculate field lists at conversion time...	**New**
Related to Database - Feature #6720: lazy hydration	**WIP**
Related to Database - Feature #3549: implement RAW-TRANSFER statement	**New**
Related to Database - Bug #7185: H2 in-memory lazy hydration	**Test**
Related to Database - Bug #7999: FWD does not honor FIELDS/EXCEPT at dynamic ...	**Review**
Related to Database - Feature #7020: always use "expanded" extent fields	**Review**

History

#1 - 04/29/2013 01:57 PM - Eric Faulhaber

- Due date changed from 07/10/2013 to 07/17/2013

- Start date changed from 06/27/2013 to 07/04/2013

#2 - 07/30/2013 05:12 AM - Eric Faulhaber

- Status changed from New to WIP

See #2095 for cases which require support.

#3 - 09/20/2013 03:07 PM - Eric Faulhaber

- Target version changed from Milestone 7 to 23

- Assignee deleted (Vadim Nebogatov)

- Status changed from WIP to New

- Due date deleted (07/17/2013)

- Start date deleted (07/04/2013)

This feature is an optimization, but is not necessary for the runtime to function properly. Moving target version accordingly.

#4 - 11/16/2016 01:21 PM - Greg Shah

- Target version deleted (23)

#5 - 07/28/2022 02:01 PM - Eric Faulhaber

- Related to Feature #6459: query features added

#6 - 08/29/2022 07:03 AM - Greg Shah

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

#7 - 09/05/2022 01:48 AM - Greg Shah

- Related to Feature #6720: lazy hydration added

#8 - 09/05/2022 01:52 AM - Greg Shah

- Related to Feature #3549: implement RAW-TRANSFER statement added

#9 - 09/05/2022 01:57 AM - Greg Shah

I was in a session the other day which was offered at the PUG Challenge 2022. This was presented by Gus, who designed major portions of the OE database engine. He documented and explained many details of the record format (the byte level representation of a record) which will be quite useful for #3549. I will post full details there. Among other things, he shared that when returning a record that has a fields clause specified, the length of a given field will be marked with a special code meaning "missing field". Each field in a record is variable size, so this code means that the field only consumes 1 byte, thus greatly shrinking the size of each record.

The approach also suggests that the 4GL has no way to obtain any further info on these fields without another query AND that the 4GL can know when a given field has been deliberately excluded. I guess that any handle or otherwise dynamic access to the buffer which tries to access a missing field will report some specific errors.

#11 - 10/19/2022 09:41 AM - Greg Shah

Code posted in #5731-399 implements partial record hydration.

#12 - 11/08/2022 08:38 AM - Constantin Asofiei

I have a (minimal) set of changes to allow EXCEPT/FIELDS at a query, with these notes:

- temp-tables do not support this feature (AFAIK this is how OE works).
- there is no failure if the app code tries to read a field which was not included in the query.
- if extent fields are tried to be added, then the entire option (EXCEPT or FIELDS) is dropped.

#5731-399 was the main change to allow this, the changes are mostly in PreselectQuery.assembleSelectClause.

#13 - 11/08/2022 09:53 AM - Constantin Asofiei

Another issue is that the records which are partially hydrated can not be cached (FastFindCache or orm.Session.cache).

#14 - 11/08/2022 11:14 AM - Constantin Asofiei

Things get tricky when you have an ignored field and you update that field - this is allowed, and OE will save the record correctly - the other ignored fields are untouched.

#15 - 11/08/2022 05:23 PM - Ovidiu Maxiniuc

Constantin Asofiei wrote:

> I have a (minimal) set of changes to allow EXCEPT/FIELDS at a query, with these notes:
>
> - temp-tables do not support this feature (AFAIK this is how OE works).

This is somehow logical. Since they are 'local' data, there is no extra cost for transporting the record's values. Unfortunately, even if H2 database functions in same process, the data is still transferred using buffers and it has to be serialized/deserialized.

- there is no failure if the app code tries to read a field which was not included in the query.

This is odd. I did not try it. The logical implementation is to fully fetch the respective at that moment. Is this OE behaviour?

- if extent fields are tried to be added, then the entire option (EXCEPT or FIELDS) is dropped.

If we work with denormalized extents, this should not be an issue in FWD, too. Keeping the EXCEPT / FIELDS options may bring some performance improvements.

> Another issue is that the records which are partially hydrated can not be cached (FastFindCache or orm.Session.cache).

I think we can implement and add bit-field (similar do BaseRecord.dirtyProps) to the cached record. If the caller request a 'lazy' field, the full record (or maybe the delta (aka: unfetched columns)) will be fetched on-the-spot.

> Things get tricky when you have an ignored field and you update that field - this is allowed, and OE will save the record correctly - the other ignored fields are untouched.

Again, this is logical. I do not need to know the old value to overwrite it. I think this works already in FWD.

The problem I see with the partial records is unique index validation. I think we should always get the (unique) index component fields, even if not specified. This way, the FastFindCache issue is solved, and we get the advantage of UniqueTracker, too.

#16 - 11/08/2022 08:18 PM - Greg Shah

> This is odd. I did not try it. The logical implementation is to fully fetch the respective at that moment. Is this OE behaviour?

In a recent presentation I saw, Gus Bjorkland (one of the lead architects/devs of the OE DB engine) explained that every field can be marked as "not present" in the record. This is done using the length value at the beginning of the field. The single byte length value has some special values, one of which means "not present". The point is that they planned for this as part of their record design. There is some defined behavior for how the 4GL acts when such fields are accessed, copied... we need to match that.

> The problem I see with the partial records is unique index validation.

Let's find out the 4GL behavior.

#17 - 11/09/2022 01:52 AM - Constantin Asofiei

Ovidiu Maxiniuc wrote:

> Constantin Asofiei wrote:
>
> > I have a (minimal) set of changes to allow EXCEPT/FIELDS at a query, with these notes:
> >
> > - temp-tables do not support this feature (AFAIK this is how OE works).
>
> This is somehow logical. Since they are 'local' data, there is no extra cost for transporting the record's values. Unfortunately, even if H2 database functions in same process, the data is still transferred using buffers and it has to be serialized/deserialized.
>
> > - there is no failure if the app code tries to read a field which was not included in the query.
>
> This is odd. I did not try it. The logical implementation is to fully fetch the respective at that moment. Is this OE behaviour?

Here I meant in FWD, not in OE. I'm adding this.

> > - if extent fields are tried to be added, then the entire option (EXCEPT or FIELDS) is dropped.
>
> If we work with denormalized extents, this should not be an issue in FWD, too. Keeping the EXCEPT / FIELDS options may bring some performance improvements.

Yes, I need to test this...

> Another issue is that the records which are partially hydrated can not be cached (FastFindCache or orm.Session.cache).

> I think we can implement and add bit-field (similar do BaseRecord.dirtyProps) to the cached record. If the caller request a 'lazy' field, the full record (or maybe the delta (aka: unfetched columns)) will be fetched on-the-spot.

I added a boolean[] parameter, but a bit-field should work.

> Things get tricky when you have an ignored field and you update that field - this is allowed, and OE will save the record correctly - the other ignored fields are untouched.

> Again, this is logical. I do not need to know the old value to overwrite it. I think this works already in FWD.

Yes, I noticed the UPDATE is build with only the changed/dirty fields.

> The problem I see with the partial records is unique index validation. I think we should always get the (unique) index component fields, even if not specified. This way, the FastFindCache issue is solved, and we get the advantage of UniqueTracker, too.

Hm... yes, I didn't consider this. Need to test.

#18 - 11/09/2022 09:35 AM - Constantin Asofiei

There is another finding: for EXCLUSIVE-LOCK, the record is always fully-fetched. In FWD, the partial record is allowed (for now) only for NONE lock-type - as you can't changed a non-locked record, you will need to do some kind of refresh to upgrade the lock (like FIND CURRENT) which will fetch the full record.

OTOH, I can't find a good way to consider extent fields for the partial record - for now I'm always reading them.

#19 - 11/09/2022 02:14 PM - Constantin Asofiei

- File 2137_20221109a.patch added

Eric/Ovidiu, please review the attached patch. The details are these:

- I couldn't find a good way to add support for the extent fields, so these are now always loaded. In OE, you can exclude/include an extent field, but not indexed - this is good (later edit: as it does not complicate things, not that it works in FWD).
- only NO-LOCK records will honor EXCEPT/FIELDS options
- if a NO-LOCK record's lock is upgraded via i.e. FIND CURRENT or otherwise is retrieved from the cache without the 'partial field information', then the full record is read from the database and the cached record's data is being set accordingly - this is to not break the 'only one Record instance must exist in the cache and referenced by buffers'.
- in OE, an EXCLUSIVE-LOCK will not honor the EXCEPT/FIELDS options, but a SHARE-LOCK (I think this is the default lock?) will honor it, even if it gets 'upgraded' when calling the setter for an 'unread' field.
- FastFindCache can not cache incomplete records

Use patch -p1 < 2137_20221109a.patch to apply it to a 3821c/14355 checkout.

#20 - 11/09/2022 02:20 PM - Constantin Asofiei

Another note: the orm.Loader.load will retrieve from the database the full records, but they will not be populated by calling ResultSet.get, only the 'partial fields' will be read, so (some) time is saved there. This needs more refactoring to allow the partial fields to be directly read from the database, considering extent fields, too.

#21 - 11/09/2022 03:43 PM - Tijs Wickardt

Constantin Asofiei wrote:

> Eric/Ovidiu, please review the attached patch. The details are these:
>
> - I couldn't find a good way to add support for the extent fields, so these are now always loaded. In OE, you can exclude/include an extent field, but not indexed - this is good (later edit: as it does not complicate things, not that it works in FWD).
> - only NO-LOCK records will honor EXCEPT/FIELDS options
> - if a NO-LOCK record's lock is upgraded via i.e. FIND CURRENT or otherwise is retrieved from the cache without the 'partial field information', then the full record is read from the database and the cached record's data is being set accordingly - this is to not break the 'only one Record instance must exist in the cache and referenced by buffers'.
> - in OE, an EXCLUSIVE-LOCK will not honor the EXCEPT/FIELDS options, but a SHARE-LOCK (I think this is the default lock?) will honor it, even if it gets 'upgraded' when calling the setter for an 'unread' field.
> - FastFindCache can not cache incomplete records
> - the orm.Loader.load will retrieve from the database the full records, but they will not be populated by calling ResultSet.get, only the 'partial fields' will be read, so (some) time is saved there. This needs more refactoring to allow the partial fields to be directly read from the database, considering extent fields, too.

Hi Constantin, thank you. The limitations are however quite present. The fact that all fields are fetched from the database doesn't concern me that

much (it's only a very small fraction of the performance hit). But for the other bullets, ideas for lazy hydration come to mind. I know that's being worked on, and forgive my ignorance if the following doesn't make sense, but what about:

1. Give each ORM data object a ByteArray, holding the raw (unserialized) data of the last fetch. Include a bitmask for (ordinal) hydrated-status yes/no per field.
2. Apply this to the FastFindCache as well
3. If all fields are hydrated, deallocate the ByteArray (set to a semantic read-only 0-size object, or even set to null)

- For compactness I leave out some bookkeeping, recid's, required keys, and more.

Could this provide a 'good enough' laziness , with positive performance impact? Mind that the raw ByteArray costs RAM, but so do the hydrated fields (less so if unboxed ints, bools, etc..., but still).

#22 - 11/09/2022 04:38 PM - Constantin Asofiei

Tijs Wickardt wrote:

> 1. Give each ORM data object a ByteArray, holding the raw (unserialized) data of the last fetch.

This is being discussed and considered, but unfortunately the JDBC drivers for H2 and PostgreSQL don't offer access to this raw state of the fetched data. Is all hidden in the JDBC implementation of the ResultSet... if we go this route, we will need changes at the JDBC driver level.

#23 - 11/09/2022 04:43 PM - Tijs Wickardt

Constantin Asofiei wrote:

> Tijs Wickardt wrote:
>
> > 1. Give each ORM data object a ByteArray, holding the raw (unserialized) data of the last fetch.
>
> This is being discussed and considered, but unfortunately the JDBC drivers for H2 and PostgreSQL don't offer access to this raw state of the fetched data. Is all hidden in the JDBC implementation of the ResultSet... if we go this route, we will need changes at the JDBC driver level.

Clear.

#24 - 11/09/2022 09:55 PM - Ovidiu Maxiniuc

Constantin Asofiei wrote:

> Eric/Ovidiu, please review the attached patch.

I like the update. The boolean arrays you talked about are already replaced by more compact BitSet.

- BaseRecord:
  - setDatum(): the conditional update of readFields should happen at the end of the method, once the data[offset] is actually set;
  - (less important) the _getReadFields() method name. I thought we try to avoid using the underscore as the prefix of a (public) method.
- SQLQuery:
  - line 760: I think this change breaks compatibility with Java 11.
  - line 778: setting hasExtents to be true when there are actually no extents forces the block at 853 to be executed. However, the result is void.
- Session: in method getImpl():639. IMHO, copying the data from one record to another at that low level is dangerous. There are flags which need to be checked and updated. I really do not know (if possible) what happens if the cached record is dirty? Will the altered fields be overwritten?
- AbstractQuery: if the collectFields() receives the already computed buffer as parameter instead of redoing this locally, do we gain some performance improvement?
- Record: does _setInteger() need to call checkIncomplete()?

#25 - 11/10/2022 01:42 AM - Constantin Asofiei

Ovidiu Maxiniuc wrote:

> - BaseRecord:
>   - setDatum(): the conditional update of readFields should happen at the end of the method, once the data[offset] is actually set;

Actually, when the setter is called, the record is already full - the changes are meant to never allow a partial record read with another lock than NO-LOCK. And you can't change a record if it has NO-LOCK.

> - (less important) the _getReadFields() method name. I thought we try to avoid using the underscore as the prefix of a (public) method.

If you add the 'get' prefix wouldn't this collide with the DMO property name?

> - SQLQuery:
>   - line 760: I think this change breaks compatibility with Java 11.

OK, I'll change it, thanks for catching it.

> - line 778: setting hasExtents to be true when there are actually no extents forces the block at 853 to be executed. However, the result is void.

Yes, that is 'by design'. I'll add a comment.

- Session: in method getImpl():639. IMHO, copying the data from one record to another at that low level is dangerous. There are flags which need to be checked and updated. I really do not know (if possible) what happens if the cached record is dirty? Will the altered fields be overwritten?

The point here is that the partial record can't be changed. Only a full record will be changed.

- AbstractQuery: if the collectFields() receives the already computed buffer as parameter instead of redoing this locally, do we gain some performance improvement?

I'll add it.

- Record: does _setInteger() need to call checkIncomplete()?

Correct, that shouldn't have been there.

#26 - 11/14/2022 10:31 AM - Constantin Asofiei

- File 2137_20221114a.patch added

Ovidiu, please review attached. This adds denormalized extent for FIELDS/EXCEPT. normalized extent is still not supported.

#27 - 11/14/2022 02:03 PM - Ovidiu Maxiniuc

Constantin Asofiei wrote:

> Ovidiu, please review attached. This adds denormalized extent for FIELDS/EXCEPT. normalized extent is still not supported.

Review of 2137_20221114a.patch:
I have a single issue in the new patch, in AbstractQuery: collectFields(): I am not sure of the intent of the inner for (int j, ... It seems to overwrite the

same props[field.propId - 1] multiple times, keeping only the last field.

#28 - 11/15/2022 08:07 AM - Constantin Asofiei

Ovidiu Maxiniuc wrote:

> Review of 2137_20221114a.patch:
> I have a single issue in the new patch, in AbstractQuery: collectFields(): I am not sure of the intent of the inner for (int j, ... It seems to overwrite the same props[field.propId - 1] multiple times, keeping only the last field.

That code is executed only for denormalized extent, and getCustomExtentFieldInfo returns the list of denormalized DMO properties - keep in mind that for denormalized mode, there is a Property.propertyID annotation emitted for each denormalized field, and this is unique, so there is no 'overwrite' happening.

#29 - 11/15/2022 09:33 AM - Ovidiu Maxiniuc

Right. I failed to notice the field is updated in inner loop.

#30 - 11/17/2022 07:35 AM - Constantin Asofiei

The 2137_20221114a.patch patch was committed to 3821c/14369, with these notes:

- only NO-LOCK records are loaded as 'partial/incomplete'. When the lock is upgraded, the partial record is always refreshed from the database, and if is cached at the ORM Session cache, that record's data is being refreshed directly.
- Partial/Incomplete records (loaded with EXCEPT/FIELDS options) can't be cached in FastFindCache.
- normalized extent fields are always loaded
- denormalized extent fields support EXCEPT/FIELDS option

#32 - 01/20/2023 07:11 AM - Constantin Asofiei

6129b/14377 has the EXCEPT/FIELDS support ported from trunk - this includes some work done by Sergey in #5731

#33 - 01/20/2023 09:51 AM - Constantin Asofiei

- % Done changed from 0 to 80

Current changes work, although normalized extent fields are always loaded. Otherwise, #6720 may touch this area again.

#34 - 03/13/2023 07:38 AM - Alexandru Lungu

- Related to Bug #7185: H2 in-memory lazy hydration added

#35 - 11/02/2023 03:54 PM - Constantin Asofiei

- Related to Bug #7999: FWD does not honor FIELDS/EXCEPT at dynamic queries added

#36 - 01/23/2024 05:03 PM - Eric Faulhaber

- Related to Feature #7020: always use "expanded" extent fields added

Files

2137_20221109a.patch	39.1 KB	11/09/2022	Constantin Asofiei
2137_20221114a.patch	48.2 KB	11/14/2022	Constantin Asofiei