Database - Feature #7020

always use "expanded" extent fields

01/05/2023 12:52 PM - Greg Shah

Status:	Review	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Ovidiu Maxiniuc	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**version:**	
vendor_id:	GCD		

Description

Related issues:

Related to Database - Feature #2137: runtime support for FIELDS/EXCEPT record... **New**

History

#1 - 01/05/2023 12:55 PM - Greg Shah

From Eric:

While you should read the whole history there, I can summarize as follows:

- We think there will be a performance benefit from denormalizing temp-table extent fields.
- When we shifted <customer_name> to use all denormalized extent fields, the change initially included temp-tables.
  - The differences in the temp-table P2O files broke the proxy Open Client generation rules, so I partially rolled back my update (see 3821c revisions 14041 and 14054).
  - I seem to recall there was another problem applying denormalization to temp-table extents, but perhaps it was only the Open Client proxies.

The problem is that denormalization changes the conversion output; specifically, the DMO interface API. This was intentional in the beginning, because we wanted to support custom naming hints, whereby each expanded column could be assigned a custom name (e.g., weekday[1] -> Monday; weekday[2] -> Tuesday, etc.).

However, in practice, this feature was never used and we are willing to remove it in order to get a consistent conversion, regardless of whether the backing implementation is denormalized extent fields, normalized (to be deprecated), or native array support at the database (e.g., for PostgreSQL). In other words, conversion always should produce DMO interfaces that look like they do when converted with the current, normalized approach. The backing implementation (denormalized vs. database-native array) should vary only in the persistence runtime, and should be decided by dialect or possibly configuration.

In scope:

- re-enabling denormalization for temp-table extent fields;
- converting DMO interfaces as if we were going to use the normalized approach;
- mapping the persistence runtime to use denormalized columns by default.

Out of scope (deferred beyond Jan 15):

- implementing database-native array types.

An alternate approach for the short term could be to consider what changes would be needed to make the proxy Open Client rules work with the denormalized temp-table P2O files generated for the modified, denormalized API, and thus avoid the larger conversion and runtime changes for now. Constantin, are you able to assess how much work this would be? However, since we do ultimately want to rationalize the conversion, this would be throwaway code, which makes this approach less appealing.

Greg, I know we had talked about the conversion rationalization being out of scope, but I hadn't remembered that this was in fact the cause of the regression with <customer_name>.

#2 - 01/05/2023 12:56 PM - Greg Shah

From Ovidiu:

I read this email several times in order to get the things right. I understand that the denormalization is going to be the standard from this point forward and the DMOs output for denormalized and optionally normalized conversion will be identical. The only thing that will change with the change will be the DDLs. It seems to be that this becomes mainly a runtime issue. As the properties will always be generated as now in normalized mode (both DMO interfaces and code occurrences using []), the backing runtime code will convert these to individual columns at some level - probably as late as Fql2SqlConverter. I guess some additional property in annotation will help.

If we get rid of the custom name (I am not sure but it seems to me that I saw it used in some places, though) we can greatly simplify the code. What I am trying to say is that now we have a denormalization at both property (Java) and column (SQL) level, but this task will eliminate the former. Since we will have a bijective relation between the extent fields and their SQL columns (field[x] <--> column_x) the Persister and Loader classes will be easily modified. Do I oversimplify things?

My second problem is that I am not familiar with Open Client proxy and its generation rules. Is this <customer_name> specific or something more general?

#3 - 01/05/2023 12:58 PM - Greg Shah

From Eric:

I think you have understood it exactly right.

> If we get rid of the custom name (I am not sure but it seems to me that I saw it used in some places, though)

If you saw the custom name feature used in some places, it may have been in test cases (or maybe Hotel?). The only customers which leverage denormalized extent fields at all are customer_x and customer_y. Neither uses hints to customize the conversion of extent field names. <customer_chui_regression_app> extensively customizes table/field names via datanames.xml and other schema hints, but not the hints specific to expanded extent fields.

> My second problem is that I am not familiar with Open Client proxy and its generation rules. Is this <customer> specific or something more general?

AFAIK, the Open Client proxy generation rules are general purpose, not specific to <customer>. Constantin, can you confirm? The rules currently are based on the assumption that the statically converted temp-table DMOs follow the "normalized" convention. The goal is to maintain this form of the DMO interface API, regardless of how extent fields are handled internally by the runtime.

One other consideration that I should have made explicit in my first note: business logic and where clause conversion also is different when the denormalized feature is enabled, in that both use references to those DMO interface expanded properties that we are proposing to eliminate. These differences will need to be eliminated as well, so that all converted business logic and FQL where clauses appear to be based on the legacy schema structure (as we do today for the normalized approach), but the underlying runtime code will need to map this to the denormalized mechanism. IIRC, some of the logic to "hide" the use of the secondary table for the normalized approach is in the FQL preprocessing logic, so there will likely be changes there, too.

> Do I oversimplify things?

I don't know yet, probably ;) Your deeper analysis will tell.

#4 - 01/05/2023 12:59 PM - Greg Shah

> AFAIK, the Open Client proxy generation rules are general purpose, not specific to <customer>. Constantin, can you confirm?

Yes, correct.

The Open Client proxy generation is the replacement of the OpenEdge "proxygen" tool which creates Java stub classes that can be used to make appserver calls from the OpenEdge "Java Open Client". Any table parameter usage and probably the dataset parameters will be dependent upon DMO generation.

#5 - 01/06/2023 10:19 PM - Ovidiu Maxiniuc

- Status changed from New to WIP

Status update.
My work ATM include:

- re-enabled denormalisation for temp-tables. I do not use this except for having a target base;
- added a new denormalized attribute in @Property annotation. It is, by default, true and it marks the properties which will be denormalized at runtime. It is meaningless for scalar properties;
- generated the new DDL for temp-tables which denormalises the properties annotated with denormalized. The name of such columns use <column-name>_<i> pattern (i = 1..extent). The dtz datatypes use <column-name>_<i>_offset for the second SQL column.

Note: momentarily there are 3 possible cases for an extent field:

- not denormalized;
- statically denormalized (at conversion time). All occurrences of this property are handled as scalar fields;
- dynamically denormalised (at runtime, based on the above attribute).

#6 - 01/06/2023 10:20 PM - Ovidiu Maxiniuc

Greg Shah wrote:

> The Open Client proxy generation is the replacement of the OpenEdge "proxygen" tool which creates Java stub classes that can be used to make appserver calls from the OpenEdge "Java Open Client". Any table parameter usage and probably the dataset parameters will be dependent upon DMO generation.

Where do I find the manual for this FWD utility? I would like the test the compatibility after my work with temp-tables is done.
Thank you!

#7 - 01/09/2023 08:39 AM - Greg Shah

- added a new denormalized attribute in @Property annotation

Please name this expanded because the "denormalized" terminology is not accurate and will be eliminated.

Where do I find the manual for this FWD utility?

ProxyGen and Open Client Usage

#8 - 01/15/2023 12:13 AM - Ovidiu Maxiniuc

- Subject changed from temp-tables should always use "denormalized" extent fields to temp-tables should always use "expanded" extent fields

I have created the 7020a branch (from 6129b) and committed r14361 which includes the changes for a first version of "expanded" extent fields.

For the moment, the attribute is set on AUTO by default and the static conversion is unaffected. This mode means

- active for temp-tables
- inactive for other (permanent) tables.

This revision adds full runtime support. There will be a next revision which will include the static conversion and hint-ing.
An example of current logic. A table defined as:

```
define temp-table den-tt
   field f1 as integer
   field e1 as integer extent 3
   field f2 as character
   field e2 as character extent 5
   field f3 as date
   field e3 as date extent 3
   field f4 as datetime-tz
   field e4 as datetime-tz extent 5
   field f5 as logical
.
```

Will be created in SQL/H2 as:

```
create local temporary table tt1 (recid bigint not null, _multiplex integer not null, _errorFlag integer
, _originRowid bigint, _datasourceRowid bigint, _errorString varchar, _peerRowid bigint, _rowState integer
, f1 integer, e1_1 integer, e1_2 integer, e1_3 integer, f2 varchar, e2_1 varchar, e2_2 varchar, e2_3 varchar
, e2_4 varchar, e2_5 varchar, f3 date, e3_1 date, e3_2 date, e3_3 date, f4 timestamp with time
 zone, f4_offset integer, e4_1 timestamp with time zone, e4_1_offset integer, e4_2 timestamp with time
 zone, e4_2_offset integer, e4_3 timestamp with time zone, e4_3_offset integer, e4_4 timestamp with time
 zone, e4_4_offset integer, e4_5 timestamp with time zone, e4_5_offset integer, f5 boolean, primary key
 (recid) ) transactional;
```

Evidently, the new records will be added using:

```
insert into
 tt1 (_errorFlag, _originRowid, _datasourceRowid, _errorString, _peerRowid, _rowState, f1, e1_1, e1_2, e1_3, f
2, e2_1, e2_2, e2_3, e2_4, e2_5, f3, e3_1, e3_2, e3_3, f4, f4_offset, e4_1, e4_1_offset, e4_2, e4_2_offset, e4
_3, e4_3_offset, e4_4, e4_4_offset, e4_5, e4_5_offset, f5, _multiplex, recid) values (?, ?, ?, ?, ?, ?, ?, ?,
?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
```

and updated using

```
update tt1 set f1=?, e1_1=?, e1_2=?, e1_3=?, f2=?, e2_1=?, e2_2=?, e2_3=?, e2_4=?, e2_5=?, f3=?, e3_1=?, e3_2=
?, e3_3=?, f4=?, f4_offset=?, e4_1=?, e4_1_offset=?, e4_2=?, e4_2_offset=?, e4_3=?, e4_3_offset=?, e4_4=?
, e4_4_offset=?, e4_5=?, e4_5_offset=? where recid=? and _multiplex=?
```

Eric, please review.

#9 - 01/16/2023 06:40 PM - Ovidiu Maxiniuc

I committed r14362.

I have a problem which I did not solved: the dynamic extent indices.
For example, for the above table and the query:

```
define variable v1 as integer init 2.
find first den-tt where e1[v1] eq 7.
```

Evidently, without the dynamic index, the SQL should be something like:

```
select * from denTt where e1_2=7;
```

I do not know how to parametrize the query to make it work as expected.

#10 - 01/18/2023 11:48 AM - Eric Faulhaber

Don't we already deal with this case in the current implementation of expanded extent fields? I may be misremembering, but I am pretty sure this issue came up before and we implemented a solution.

#12 - 01/18/2023 12:19 PM - Eric Faulhaber

- Subject changed from temp-tables should always use "expanded" extent fields to always use "expanded" extent fields

#13 - 01/18/2023 04:22 PM - Eric Faulhaber

Code review 7020a/14361-14362:

Nice work! Thank you for turning it around in such a short time.

TBH, it is hard to know whether the generated SQL in the FQL-SQL generator, loader, persister is correct from just a review. I didn't see anything wrong there, but I will need to rely on testing to confirm the rework is without regression. How much testing have you done so far?

Please rename TempTableHelper method addcolumn to addColumn (camelCase).

The new method getIndexComponents in RecordMeta is not used, AFAICT.

On the dynamic index issue, we will have the variable's value at runtime. Isn't this enough to resolve the correct extent field "element" (i.e. discrete column) at SQL generation, or during FQL preprocessing, if needed? Unless the dynamic index is a field reference in the current buffer, the where clause conversion should have refactored v1 (in the example above) to a query substitution parameter, for which we will have the substitution value with which to execute the query. In fact, I thought we already do this, as we have to deal with the same dynamic index syntax already today, even though we refactor the extent field into separate DMO properties currently.

Generally: in comments, please don't use the terms "normalized" when referring to the extent field notation of the legacy code. This is confusing, as "normalized" already was a misleading term when I started using it, and it was meant to refer to the structure of the tables in SQL. I know I probably encouraged this confusion in my description of this issue's purpose (sorry about that), but let's correct it now, so the code isn't forever confusing to someone new coming along:

- I think "[legacy] array notation" is probably the right term to use when referring to the 4GL syntax for referencing a specific extent field element (e.g., contact-info.address[1]). I refer to "legacy" as optional, because we may want to differentiate this from any code which we may soon add to support native database arrays.
- We've already agreed on "expanded" as the term to replace "denormalized" (the latter also was a poor choice on my part, way back when), and it looks like you've already represented it this way in the code/comments. There shouldn't really be a notion of "expanded" in terms of the converted code, so this term pretty much applies only to the generation of and runtime use of SQL DML and DDL/schema.
- Let's not refer to approaches as old/new, unless using this information as a temporary guide to identify code that will be removed/modified shortly, as we complete the transition to use expanded columns or native arrays only. My point is that I don't want the temporal notions of older/newer to linger in the code forever, since at some point soon, we will be finished with this transition and the only approaches that matter will be the ones encoded at the end.

I understand the need to continue the support for the "normalized" schema for now, until we can transition all projects to the expanded or native array approaches. Expanded should be the default for BOTH temp-tables and persistent tables, and we should have a temporary configuration option to continue to use the "normalized" schema. Going forward, we probably will handle the native array support through dialects, and perhaps a conversion configuration override.

These defaults should be reflected in the Property.expanded annotation, and whatever downstream code that change affects. The new AUTO/DEFAULT setting assumes expanded for temporary tables, but normalized for permanent tables. We do not want to default permanent tables to a normalized schema, as this will result in new projects coming online with the wrong schema. The defaults need to encourage the extinction of the normalized approach.

#14 - 01/18/2023 07:36 PM - Ovidiu Maxiniuc

Eric Faulhaber wrote:

> Code review 7020a/14361-14362:
> Nice work! Thank you for turning it around in such a short time.

Thank you!

> TBH, it is hard to know whether the generated SQL in the FQL-SQL generator, loader, persister is correct from just a review. I didn't see anything wrong there, but I will need to rely on testing to confirm the rework is without regression. How much testing have you done so far?

Not much, Several of my targeted testcases. I usually do a smoke-test using the hotel project, but it does not use _temp tables with extent fields to be expanded.

> Please rename TempTableHelper method addcolumn to addColumn (camelCase).

Done. This happens accidentally. It's strange that I did not notice in my own clean-up/review.

> The new method getIndexComponents in RecordMeta is not used, AFAICT.

It was not related to this task. I will remove it from the branch.

> On the dynamic index issue, we will have the variable's value at runtime. Isn't this enough to resolve the correct extent field "element" (i.e. discrete column) at SQL generation, or during FQL preprocessing, if needed? Unless the dynamic index is a field reference in the current buffer, the where clause conversion should have refactored v1 (in the example above) to a query substitution parameter, for which we will have the substitution value with which to execute the query. In fact, I thought we already do this, as we have to deal with the same dynamic index syntax already today, even though we refactor the extent field into separate DMO properties currently.

I will create a testcase to test the solution in denormalized case.

> Generally: in comments, please don't use the terms "normalized" when referring to the extent field notation of the legacy code. This is confusing, as "normalized" already was a misleading term when I started using it, and it was meant to refer to the structure of the tables in SQL. I know I probably encouraged this confusion in my description of this issue's purpose (sorry about that), but let's correct it now, so the code isn't forever confusing to someone new coming along:
>
> - I think "[legacy] array notation" is probably the right term to use when referring to the 4GL syntax for referencing a specific extent field element (e.g., contact-info.address[1]). I refer to "legacy" as optional, because we may want to differentiate this from any code which we may soon add to support native database arrays.
> - We've already agreed on "expanded" as the term to replace "denormalized" (the latter also was a poor choice on my part, way back when), and it looks like you've already represented it this way in the code/comments. There shouldn't really be a notion of "expanded" in terms of the converted code, so this term pretty much applies only to the generation of and runtime use of SQL DML and DDL/schema.
> - Let's not refer to approaches as old/new, unless using this information as a temporary guide to identify code that will be removed/modified shortly, as we complete the transition to use expanded columns or native arrays only. My point is that I don't want the temporal notions of older/newer to linger in the code forever, since at some point soon, we will be finished with this transition and the only approaches that matter will be the ones encoded at the end.

Thank you for clarification. I will update the comments and javadocs.

> I understand the need to continue the support for the "normalized" schema for now, until we can transition all projects to the expanded or native array approaches. Expanded should be the default for BOTH temp-tables and persistent tables, and we should have a temporary configuration option to continue to use the "normalized" schema. Going forward, we probably will handle the native array support through dialects, and perhaps a conversion configuration override.

The solution with AUTO was so that I could differentiate at runtime, between the _temp and permanent tables so that the update would work for existing code without the need to reconvert the whole application. The DDLs are already generated and database already imported with possible denormalized structure. This solution avoid immediate interferences here. OTOH, the DDLs and SQL for _temp tables are all dynamically generated at runtime so it works on exiting converted code, only if the extents were not denormalized. If they were, the getters and setters for the properties are changes and their columns are already stored in the primary table.

> These defaults should be reflected in the Property.expanded annotation, and whatever downstream code that change affetc. The new AUTO/DEFAULT setting assumes expanded for temporary tables, but normalized for permanent tables. We do not want to default permanent tables to a normalized schema, as this will result in new projects coming online with the wrong schema. The defaults need to encourage the extinction of the normalized approach.

I would like to keep it this way for a little, until the conversion is "expanded"-aware. At that moment, the annotations will clarify the exact solution for the extent fields. As noted above, for _temp tables, this can be altered with each execution, but for permanent tables, the reconversion and re-import of the database is required.

#15 - 01/18/2023 07:51 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> I would like to keep it this way for a little, until the conversion is "expanded"-aware. At that moment, the annotations will clarify the exact solution for the extent fields. As noted above, for _temp tables, this can be altered with each execution, but for permanent tables, the reconversion and re-import of the database is required.

Fair enough, ok for now. But let's not forget to change this default behavior for FWDv4. Customers will need to re-convert their applications anyway, due to all the other conversion changes. At that time, for anyone currently using the deprecated schema, we will need to provide a means to refactor their database schema in place. But that is beyond the scope of this task, so we can discuss it at another time.

#16 - 01/20/2023 12:19 AM - Ovidiu Maxiniuc

Eric Faulhaber wrote:

> On the dynamic index issue, we will have the variable's value at runtime. Isn't this enough to resolve the correct extent field "element" (i.e. discrete column) at SQL generation, or during FQL preprocessing, if needed?

Theoretically, yes. The lookup is not trivial, the generated SQL is not reusable (except for similar queries which have the same index for the respective property). This should be done during FQL preprocessing. The converter is parameter-independent because its output is designed to be cached and reusable.

Unless the dynamic index is a field reference in the current buffer, the where clause conversion should have refactored v1 (in the example above) to a query substitution parameter, for which we will have the substitution value with which to execute the query. In fact, I thought we already do this, as we have to deal with the same dynamic index syntax already today, even though we refactor the extent field into separate DMO properties currently.

Unfortunately, as resulting from my simple test the answer is negative. Attempting to execute a query with dynamic index as described in #7020-9 will fail like this:

```
org.h2.jdbc.JdbcSQLSyntaxErrorException: Syntax error in SQL statement "
SELECT [...]
FROM     TT1 DENTT_1_1_0_
WHERE    DENTT_1_1_0_._MULTIPLEX = ? (DENTT_1_1_0_.E1?[*] != 1 OR DENTT_1_1_0_.E1? IS NULL) AND [...]
ORDER BY DENTT_1_1_0_._MULTIPLEX DESC, DENTT_1_1_0_.RECID DESC
```

The occurrence is simply emitted as DENTT_1_1_0_.E1? (twice) with the parameter added to parameter-list. Notice the [*] injected by H2 when reporting the error.

The FqlToSqlConverter complains about the possible issue, twice (Failed to process extent ALIAS. and Property 'e1' not found to extract column name), but at this time is already too late.

I remember we encounter some time ago this issue. Apparently we have not implemented it yet. See #2134-7 and following. This is actually the only related task I could find.

I think we can fix this issue. Another problem is the self-referencing in constructs like this:

```
find last den-tt where e1[f1] ne k.
```

In this very simple case the conversion prefers to grab all data from SQL and handle it locally:

```
new FindQuery(denTt, (String) null, () -> isNotEqual(denTt.getE1(minus((denTt.getF1()), 1)), k), "denTt.recid
asc").last();
```

This particular case should correctly though, even if with a big performance hit, especially for big table. If the table is really big, this is not feasible as no records are filtered out.

I have committed my latest changes and rebase the branch to 6129b. The current revision of 7020a is 14377.

#17 - 01/20/2023 03:41 AM - Constantin Asofiei

I've tested 7020a/14377 and there is an abend with a customer's application.

#18 - 01/20/2023 02:27 PM - Ovidiu Maxiniuc

Thank you for doing the tests.
May I have some details on this abend? An error message or a stack trace / log file?

I do not have a recent conversion source/binaries of the customer's application and I am not aware of a repository for this. Where can I get the pre-built customer project so I can test myself?

#19 - 01/23/2023 07:42 PM - Ovidiu Maxiniuc

- % Done changed from 0 to 80

Thank you Constantin for the link to wiki. I was able to identify the cause of the NPE. When retrieving meta information on indices, FWD assumed the expanded properties are also pushed to the end of record. But that is not the case: they are expanded in-place.

I committed the fix in 14384 of 7020a. The branch was also rebased to r14380 of 6129a.

#21 - 02/20/2023 08:26 PM - Ovidiu Maxiniuc

- Status changed from WIP to Review

- % Done changed from 80 to 100

I attempted to rebase 7020a to trunk, but that proved to be a very time consuming task, practically duplicating Constantin's work of last week.
My solution was to create a new branch 7020b from current trunk and reapply the cherry-picked changes from 7020a. The downside is that the half dozen commits are now flattened.

Committed revision 14485.
Eric, please review.

#23 - 09/15/2023 10:27 AM - Greg Shah

Eric: Please review.

#25 - 01/23/2024 02:04 AM - Alexandru Lungu

Ovidiu, can you rebase 7020b? I will like to pick it up and do some review / tests / profiling with it.

#26 - 01/23/2024 05:03 PM - Eric Faulhaber

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

#27 - 01/24/2024 09:28 PM - Eric Faulhaber

Constantin, besides me taking forever to review 7020b, we still have the issue with the Client Proxy generation blocking this task, don't we? What will it take to resolve that?