Database - Feature #3758

misc database features

10/21/2018 02:41 PM - Greg Shah

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Eric Faulhaber	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**version:**	
vendor_id:	GCD		

Description

Related issues:

Related to Database - Feature #3813: misc DB features part deux	**Closed**
Related to Database - Feature #3549: implement RAW-TRANSFER statement	**New**

History

#1 - 10/21/2018 02:45 PM - Greg Shah

Functions

- DBTASKID
- RECORD-LENGTH

Attributes

- SERIALIZE-HIDDEN

Methods

- BUFFER:FIND-CURRENT
- BUFFER/TT:READ-JSON
- BUFFER/TT:WRITE-JSON
- QUERY:FIRST-OF
- QUERY:LAST-OF

#2 - 11/01/2018 12:31 PM - Eric Faulhaber

- Status changed from New to WIP

- Assignee set to Eric Faulhaber

Created task branch 3758a.

#3 - 11/25/2018 09:25 AM - Greg Shah

- Related to Feature #3813: misc DB features part deux added

#4 - 12/09/2018 11:05 PM - Eric Faulhaber

- % Done changed from 0 to 30

Task branch 3758b contained conversion support and partial/stubbed runtime support for DBTASKID, RECORD-LENGTH builtin functions; FIND-CURRENT, READ/WRITE-JSON methods. It passed conversion testing, was merged to trunk, and committed as revision 11296. Runtime testing was not performed, since the changes were additive only.

#5 - 12/09/2018 11:55 PM - Eric Faulhaber

The SERIALIZE-HIDDEN attribute runtime support was not implemented at this time. See #3750-41.

#6 - 01/02/2019 11:33 AM - Greg Shah

Eric: Please summarize what needs to be done for this task.

#7 - 01/02/2019 03:01 PM - Eric Faulhaber

Greg Shah wrote:

> Eric: Please summarize what needs to be done for this task.

DBTASKID Function
Full runtime implementation is needed. Several variants of ConnectionManager.dbTaskId are stubbed, currently call UnimplementedFeature.missing method. Currently, there is no infrastructure to track a current transaction identifier, so TransactionManager updates will be needed. Since the use of this function should be very rare, the overhead of maintaining a transaction identifier must absolutely be minimized.

RECORD-LENGTH Function
Full runtime implementation is needed. BufferImpl.recordLength is stubbed, currently calls UnimplementedFeature.missing method. Implementation should be added to RecordBuffer class. The following looks useful for the implementation:
https://documentation.progress.com/output/ua/OpenEdge_latest/index.html#page/gsdbe%2Fformulas-for-calculating-field-storage.html%23

SERIALIZE-HIDDEN Attribute
Not needed at this time, as documented previously.

BUFFER:FIND-CURRENT Method
Needs to be wired to existing, backing support in the persistence runtime. Did not fit with common implementation used by BUFFER:FIND-{FIRST|NEXT|PREVIOUS|LAST}, which all delegate to the BufferImpl._find worker method, because that method converts a dynamic query predicate. This is not appropriate for FIND-CURRENT, so some refactoring is needed.

BUFFER/TT:{READ|WRITE}-JSON Methods
Runtime implementation is needed. I began implementation based on the similar XML methods, but mostly just to the extent needed to support conversion. That is, I created a JsonData interface, which Buffer extends and BufferImpl implements. However, the implementation is stubbed: the less specific BufferImpl.{read|write}Json methods delegate to more specific variants with appropriate default parameters, but the most specific worker method is not implemented. It calls UnimplementedFeature.missing.

QUERY:{FIRST|LAST}-OF Methods
Fully implemented.

#8 - 01/16/2019 04:04 PM - Ovidiu Maxiniuc

Status & notes:

> **BUFFER:FIND-CURRENT Method**
> Needs to be wired to existing, backing support in the persistence runtime. Did not fit with common implementation used by BUFFER:FIND-{FIRST|NEXT|PREVIOUS|LAST}, which all delegate to the BufferImpl._find worker method, because that method converts a dynamic query predicate. This is not appropriate for FIND-CURRENT, so some refactoring is needed.

Implemented. However, the reference does differ from the observed behaviour. I also added support for the related CURRENT-CHANGED attribute. Where should I commit? My update is based on 3750a.

> **DBTASKID Function**
> Full runtime implementation is needed. Several variants of ConnectionManager.dbTaskId are stubbed, currently call UnimplementedFeature.missing method. Currently, there is no infrastructure to track a current transaction identifier, so TransactionManager updates will be needed. Since the use of this function should be very rare, the overhead of maintaining a transaction identifier must absolutely be minimized.

ETA: 25%. I did some research here. The documentation is scarce. There is some support in TransactionManager (see nextTransId and BlockDefinition.transId), but this does not seem correct. Here are some notes from my investigations:

- transaction IDs are independent for each database. They increment separately, two or more database can have the same transaction ID at different moments in time or even at the same time;
- apparently, the next id is persistent for each database (probably in a VST field);
- the transaction ID is only assigned when the first buffer is locked in EXCLUSIVE in a TRANSACTION block, not when the block is pushed on stack;
- even if a database is connected, the transaction ID is not set for it, if no buffers are EXCLUSIVE locked for respective database;
- the transaction ID is set at the moment of EXCLUSIVE lock. It does not matter whether the record is actually updated or otherwise touched.

#9 - 01/17/2019 08:02 AM - Ovidiu Maxiniuc

Continuation:
The DBTASKID function is related to _trans VST. However, the relation is not 1:1. If, in a transaction, any record from a database is involved (updated OR FIND, regardless of the lock level - and this includes the _Trans VST, too) then the task id is incremented and the new value is returned by DBTASKID function. Otherwise (all records of the database are not accessed in any way OR just read access to buffers loaded before the transaction started) the _Trans._Trans-Num of the database is NOT incremented and DBTASKID function returns ? (unknown value).

#10 - 01/18/2019 09:29 AM - Greg Shah

> Where should I commit? My update is based on 3750a.

Please use 3750b. It is based on trunk 11298 (which is 3750a).

#11 - 01/21/2019 01:03 PM - Ovidiu Maxiniuc

3750b was updated. The list of changes in 11303 is the following:

- added support for CURRENT-CHANGED, FIND-CURRENT;
- added support for DBTASKID ABL function;
- improved _Trans VST metadata support;
- fixed locking in FIND_* methods;
- fixed DBPARAM function.

Notes related to this update:
The new implementation is largely based on the old code of TransactionUpdater, but the data processed is rather different because the transaction id are generated at different moment, not at the beginning of the transaction block. It is 'flushed' to H2 database only if there is a query on _Trans VST, otherwise the data structure used by DBTASKID function is stored in memory, so it's fast. The DBTASKID works fine now.

When a _Trans buffer is needed in a query, the internal data is requested to be 'flushed' to H2 meta database so it could be accessed in SQL form by the about-to-be-executed-query. The problem is that, because the fresh _Trans data, although saved at this moment, is not really committed until the end of transaction, the queries on this VST table will not be able to access them yet. The data is available only after the transaction ends and commit is performed. Needless to say that at that moment, the data is invalid as the transaction has actually ended.

I wonder how can we force the commit of the in-memory data when it is needed in SQL form? Initially I thought that the auto-commit would be a solution, but this only works outside the transactions. As a test I've manually called commit on the persistence (and then reopened the SQL/Hibernate transaction - to let things apparently unchanged). It worked for my isolated testcase, but this is not going to work for a whole application.

An interesting note: you cannot get a lock on _Trans buffers. I 'successfully' acquired EXCLUSIVE-LOCKS on same record from multiple ABL clients. In fact, I think, it is not possible to have the lock on neither of VS tables (this assertion remains to be tested). As I understand them now, they are not real rows, but pieces of information extracted in real-time from ABL db engine.

#12 - 01/21/2019 02:53 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> The new implementation is largely based on the old code of TransactionUpdater, but the data processed is rather different because the transaction id are generated at different moment, not at the beginning of the transaction block.

What is the actual requirement in this regard? When does the transaction id need to be generated? I understand that the single id we were generating before is not appropriate; we need a unique transaction id per connected database, right?

> It is 'flushed' to H2 database only if there is a query on _Trans VST, otherwise the data structure used by DBTASKID function is stored in memory, so it's fast. The DBTASKID works fine now.

It was not so much the speed of the DBTASKID function I was concerned with; this will be used rarely. It's good that it is fast, but it is really the overhead of the transaction id management that I am concerned with. I think we have added more overhead than necessary with this implementation. If I read the code correctly, you are checking whether the transaction id is set every time we load a record into a buffer and any time we call a setter on a DMO.

Don't we potentially need to update transaction ids on the following events?

- when starting a full transaction;
- when ending a full transaction;
- when connecting a database (if a transaction is active);
- when disconnecting a database (if a transaction is active).

This should result in far fewer checks (which are somewhat expensive in the multiple levels of lookup they do) than each time a record is set into a buffer and every time a DMO setter is called.

Are there any other events that would require a check? Is the transaction id changed at a subtransaction boundary, or just full transaction?

I'm not thrilled that so much database code has moved into TM. Why was it necessary to dismantle the listener model? The point of this was to provide a looser coupling between the TM and the VST implementation. Perhaps events need to be fired from different (or more) locations now, but I think the listener model is still appropriate.

> When a _Trans buffer is needed in a query, the internal data is requested to be 'flushed' to H2 meta database so it could be accessed in SQL form by the about-to-be-executed-query. The problem is that, because the fresh _Trans data, although saved at this moment, is not really committed until the end of transaction, the queries on this VST table will not be able to access them yet. The data is available only after the transaction ends and commit is performed. Needless to say that at that moment, the data is invalid as the transaction has actually ended.
>
> I wonder how can we force the commit of the in-memory data when it is needed in SQL form? Initially I thought that the auto-commit would be a solution, but this only works outside the transactions. As a test I've manually called commit on the persistence (and then reopened the SQL/Hibernate transaction - to let things apparently unchanged). It worked for my isolated testcase, but this is not going to work for a whole application.

If we currently are managing the metadata databases using application-level transactions, that is a bug. Since, as you noted, VST records cannot be locked or changed by application code, there is no point in applying application-level transactions to them. The Persistence$Context object for the metadata database should have transactions that are independent of the application's transactions for the primary and temp-table databases. Changes to the meta_trans table (and meta_connect, etc.) must be committed independently, so that information is available to all contexts regardless of application-level transaction scopes.

I don't think the flush point you added to Persistence.load is appropriate (where you have the TODO asking if its appropriate). load is called after the query already has executed; at that point, we're just retrieving the found DMO, so doing the flush then is too late. Did you find that the metadata flush in RecordBuffer.flush was not enough? This represents the moment when we have determined a record (any record) would be flushed to the database.

Outside of the _trans changes, the rest of the update looks good to me.

#13 - 01/22/2019 06:51 AM - Ovidiu Maxiniuc

Eric Faulhaber wrote:

> What is the actual requirement in this regard? When does the transaction id need to be generated? I understand that the single id we were generating before is not appropriate; we need a unique transaction id per connected database, right?

The transaction id is generated only when a database access occur (ie: a FIND, or a field update is sufficient). If there are no database accesses in a TRANSACTION block the transaction ID is not generated and DBTASKID returns unknown for at that moment. Each database keeps a transaction counter. It will be incremented and assigned for any user that access it in a valid transaction. Two different databases can have transactions with the same transaction-id for same or for different users at the same time.

> It was not so much the speed of the DBTASKID function I was concerned with; this will be used rarely. It's good that it is fast, but it is really the overhead of the transaction id management that I am concerned with. I think we have added more overhead than necessary with this implementation. If I read the code correctly, you are checking whether the transaction id is set every time we load a record into a buffer and any time we call a setter on a DMO.

Yes, this is true. But all happens in memory, and the accesses are hash-mapped. The overhead is caused only by two things:

- the synchronization on the data member;
- the access to SecurityManager 's context to identify the session id.

These are far faster than any H2 access. The H2 access is only effectuated if the _Trans table is accessed in read mode. The ABL programmer does not have write access to (this) VST. When _Trans is queried, I try to flush (or better said synchronize) the in-memory content for respective database with H2. If no such operations are used, then all happens in-memory.

> Don't we potentially need to update transaction ids on the following events?
>
> - when starting a full transaction;
> - when ending a full transaction;
> - when connecting a database (if a transaction is active);
> - when disconnecting a database (if a transaction is active).

As noted above, the IDs are not generated when the transaction block is pushed to (popped from) stack and the transaction counters are independents for each database. If a database is connected during a transaction it will assign a new transaction-id the first time a record is fetched or updated. If that is the first ever transaction, the counter will start at 1, otherwise will increment the counter to N+1, where N is the id of a possible still-running transaction in another connection/user.

> This should result in far fewer checks (which are somewhat expensive in the multiple levels of lookup they do) than each time a record is set into a buffer and every time a DMO setter is called.

I know, but I based my implementation on the results and observations done on native ABL.

> Are there any other events that would require a check? Is the transaction id changed at a subtransaction boundary, or just full transaction?

Subtransactions are not special here. If the current transaction has an ID then it will be kept during the subtransaction and until the parent transaction ends. If the subtransaction is the first place the db is accessed, the parent transaction gets the ID and keeps it until it ends regardless whether there are no more db accesses in it after the subtransaction block ended. It looks just like any other normal block in the transaction block. I.e.: the _Trans record is dropped (and dbtaskid returns ?) when the parent transaction block ends.

> I'm not thrilled that so much database code has moved into TM. Why was it necessary to dismantle the listener model? The point of this was to provide a looser coupling between the TM and the VST implementation. Perhaps events need to be fired from different (or more) locations now, but I think the listener model is still appropriate.

I preferred that because the data was there. A listener mechanism is not needed. However, the de-coupling TM and the VST persistence is good. I will fix this. In fact this is required since the TM is used at conversion time, too.

> If we currently are managing the metadata databases using application-level transactions, that is a bug. Since, as you noted, VST records cannot be locked or changed by application code, there is no point in applying application-level transactions to them. The Persistence$Context

> object for the metadata database should have transactions that are independent of the application's transactions for the primary and temp-table databases. Changes to the meta_trans table (and meta_connect, etc.) must be committed independently, so that information is available to all contexts regardless of application-level transaction scopes.

I think we do. For the moment, I do not know how to do this. I noticed that some of the management of _meta tables (not sure if VSD only) is performed using native H2 calls using executeSQL() calls. This could be a solution but I do not like the idea very much, because, even if we are using now only H2 for meta and temp databases, the native code will get us stuck with H2 and we are losing flexibility here.

> I don't think the flush point you added to Persistence.load is appropriate (where you have the TODO asking if its appropriate). load is called after the query already has executed; at that point, we're just retrieving the found DMO, so doing the flush then is too late. Did you find that the metadata flush in RecordBuffer.flush was not enough? This represents the moment when we have determined a record (any record) would be flushed to the database.

Yes, that is a remaining of my work for finding the most appropriate the places to check and flush in-memory data to H2. I spent a lot of time moving these calls so that the impact on performance to be minimum. Probably I failed to notice it in the final clean-up. I will review and re-test this occurrence and most likely I will remove the call.

> Outside of the _trans changes, the rest of the update looks good to me.

Thank you very much for the review.

#14 - 01/28/2019 02:47 AM - Eric Faulhaber

3750b/11317 adds basic runtime support for the {READ|WRITE}-JSON methods. Basic JSON import and export work with some limitations and a few known issues:

- the following types are supported:
  - types assignable from int64
  - types assignable from Text
  - types assignable from date
  - decimal
  - logical

I need to look at what needs to be done to handle the other data types that can be used with temp-tables.

I've tested the following:

- integer

- character
- decimal - the format (precision) does not always match ABL output
- date - the format does not match ABL output; we produce MM/DD/YY text, while ABL outputs YYYY-MM-DD
- logical

Unformatted JSON is identical to ABL's output. Formatted JSON does not look identical (some differences in line breaks and indents).

Like with the XML export, not all options are supported for JSON export yet. Currently, we support FILE and LONGCHAR target types, but not yet STREAM, STREAM-HANDLE, MEMPTR, JsonArray, or JsonObject. We currently ignore the omit-initial-values, omit-outer-object, and write-before-image options. I believe the last two are related to datasets.

Some refactoring of the JSON (and XML) import/export code will be needed, once we have a better grasp on the requirements for dataset integration, as the APIs currently operate on an instance of TemporaryBuffer.

Despite the format issues, the data imports correctly from the given output and iterating through the temp-tables and DISPLAY-ing the records gives the same results.

As part of these changes, I did some refactoring of the XML import/export runtime code. Hopefully, I did not break that, but some regression testing is required to be sure.

#15 - 02/05/2019 09:23 PM - Ovidiu Maxiniuc

Eric,
Do we need to find the exact formula for RECORD-LENGTH?
I have implemented a formula that works well for a small amount of fields of a record. However, when the number of fields in the record increases, ABL additionally add supplementary increments of 5 bytes. I could not find a prognosis when they would be added.

#16 - 02/05/2019 10:40 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> Do we need to find the exact formula for RECORD-LENGTH?
> I have implemented a formula that works well for a small amount of fields of a record. However, when the number of fields in the record increases, ABL additionally add supplementary increments of 5 bytes. I could not find a prognosis when they would be added.

Did you base your implementation on
https://documentation.progress.com/output/ua/OpenEdge_latest/index.html#page/gsdbe%2Fformulas-for-calculating-field-storage.html%23?

I'll review tomorrow...

#17 - 02/06/2019 08:21 AM - Ovidiu Maxiniuc

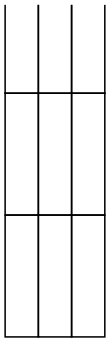
Eric Faulhaber wrote:

> Did you base your implementation on https://documentation.progress.com/output/ua/OpenEdge_latest/index.html#page/gsdbe%2Fformulas-for-calculating-field-storage.html%23?

Yes, that table provides a rough estimation for a single field only. However, as the number of fields in a record increases, 4GL adds extra bytes. After further investigations I was able to come up with some patterns:
For example: for a record with a single logical extent, all components set to unknown value. Here is the variation table:

extent	size of record	notes
1	20	computed as 15 + 4 +1
2-15	21-34	+1 for each component
16	40	+1 +additional 5 bytes are added
17-31	41-55	+1 for each component
32	58	+1 +additional 2 bytes are added
33-47	59-73	+1 for each component
48	76	+1 +additional 2 bytes are added

etc..
But that is not all. The size also depends on the number of fields in the record. Let's add some simple logical fields to initial with extent of 1.



etc..
So, it seems that, there is a threshold at each 16 components/ fields. The fact that for simple fields the first one is at 13 implies that the extra constant 15 bytes are occupied by hidden 2 fields that forms some kind of header (most likely one of them is the recid/rowid). It is important to note that at first threshold, 5 bytes are added, while at the subsequent only 2.

> I'll review tomorrow...

I have not committed yet, I don't know where to: 3750b or the new 3898a? The code is really additive so it's safe. I will update it in the light of my new discoveries and let you know when ready.

#18 - 02/06/2019 09:16 AM - Eric Faulhaber

3750b

#19 - 02/06/2019 09:50 AM - Greg Shah

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

#20 - 02/06/2019 09:52 AM - Greg Shah

Please see #3549 for my previous research into the layout of a 4GL serialized database record. It has findings that are overlapping with your work. As you can imagine, one important use of RECORD-LENGTH would be to allocate a buffer that can hold the entire serialized record. I believe that the RECORD-LENGTH returns the exact size of what would be serialized.

#21 - 02/06/2019 06:05 PM - Ovidiu Maxiniuc

I added (full) implementation of RECORD-LENGTH to 3750b. Committed revision 11340.
The solution central point is in BufferImpl.recordLength() which is computed the extra bytes as described in note-17. However, each individual field

needs to report the size of data it holds so I added a new method in BDT: getSize(). I have done extensive testing to identify how each of the fields are serialized. The main set of tests were saved as testcases/uast/raw_transfer/record-length.p.

Please review.

#22 - 02/19/2019 12:43 PM - Greg Shah

Code Review 3750b Revision 11340

I'm good with the changes. I like the approach with the BDT getSize(). When we implement the fully compatible RAW-TRANSFER we will do it the same way (putting the type-specific serialization there).

#23 - 02/19/2019 12:44 PM - Greg Shah

Can this task be closed?

#24 - 02/19/2019 01:14 PM - Ovidiu Maxiniuc

Greg Shah wrote:

> Can this task be closed?

RECORD-LENGTH: I think I covered all datatypes which can be serialized. We cannot test this at this time.
DBTASKID: All my tests were successful for this function. There is a clear delimitation from the related _Trans VST whish is not 100% done, but that is being fixed with other VSTs.
BUFFER:FIND-CURRENT: Fully functional.

I believe Eric handled the others. In conclusion, yes, it can be closed.

#25 - 02/19/2019 02:14 PM - Greg Shah

- % Done changed from 30 to 100

- Status changed from WIP to Closed

#26 - 11/03/2019 06:25 AM - Greg Shah

Ovidiu: The DBTASKID() is marked as runtime stubs in gap marking. I think this is incorrect. Based on the comments above, it possibly should be "full" though there is some work on the _trans table which is not clear whether or not it impacts DBTASKID() behavior.

#27 - 11/04/2019 08:34 AM - Ovidiu Maxiniuc

Indeed, the DBTASKID should be marked as having full implementation. Where do I commit this?

#28 - 11/06/2019 09:16 AM - Greg Shah

Ovidiu Maxiniuc wrote:

> Indeed, the DBTASKID should be marked as having full implementation. Where do I commit this?

4069a, thanks.