Database - Bug #5056

Temp-tables and unique indexes

12/22/2020 01:19 PM - Ovidiu Maxiniuc

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

Description

History

#1 - 12/22/2020 01:26 PM - Ovidiu Maxiniuc

This is a discussion started by email.

On 12/21/20 6:50 PM, Ovidiu Maxiniuc wrote:

> Eric,
>
> Do you recall why we dropped the unique constraint from TEMP-TABLES?
> I did this on purpose in revision 11347.1.310 on 9/1/20 2:49 AM. I even left a comment how to re-enable it back (see line 415 in TempTableHelper.java), but I really do not remember what caused this decision. There are some insights in #4832 but I cannot get the full picture.
>
> I would like to have it back because validation is based on SQL and, if unique indexes are disabled, SQL is unable to detect duplicate keys.

On 12/22/20 2:42 AM, Eric Faulhaber wrote:

> Ovidiu,
>
> The method RecordNursery.makeVisible is called by queries which may need to find new, partially updated, but not yet fully valid records. That is, the 4GL will update a table's indices with newly created records before those records are necessarily flushed and formally validated, such that they will come up in results (e.g., for a FIND) which are found by walking those indices.
>
> For persistent records, we use the dirty database to simulate these early, invalid records being added to a shared table. Since temp-tables don't use the dirty database, we have to be able to persist not-yet-validated records to a temp-table, so that they can show up in those results. We disabled the SQL-level unique constraint on temp-tables to permit this to happen. A SQL version of a legacy unique index is created, it is just not marked unique. When these records eventually are validated, the legacy unique indices are taken into account in composing the query, but there is no use of a SQL-level unique index as part of the validation.
>
> > I would like to have it back because validation is based on SQL and, if unique indexes are disabled, SQL is unable to detect duplicate keys.
>
> We no longer rely on a micro-savepoint to validate a new temp-table record. All temp-table record validation in the Validation class is done by query now. Putting the unique keyword back in temp-table create index statements will severely break legacy query behavior as described above.

On 12/21/20 7:58 PM, Ovidiu Maxiniuc wrote:

> My problem is that a temp-table keeps accumulating newly-created 'validated' records which break the unique index constraint. I reactivated the unique indexes for temp-tables and the problem is fixed.
>
> However, since this will break legacy queries I will roll that change back. But I need t know when whether a record passes validation. At this moment the code looks like this:

```
boolean validated = false;
try
{
   validated = buffer().validate(false);
}
catch (ValidationException e)
{
   // ignore, no 'expected' exception will be thrown
}

// was there an unique conflict with this record?
if (!validated)
   ...
```

I think this kind of code is used in a few other places. What do you mean by *"record validation in the Validation class is done by query"*?

The code above will eventually call validateMaybeFlush() which will create a new Validation object and call validateMaybeFlush() on it.

Do I need to manually check the uniqueness ?

#2 - 12/22/2020 07:02 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> My problem is that a temp-table keeps accumulating newly-created 'validated' records which break the unique index constraint.

This problem probably is caused by this code in Validation.validateUniqueIndices(BitSet):

```
         if (flush || buffer.isAutoCommit())
         {
            // validation will be done by the database as part of the flush (for persistent tables)
            flush();
         }
```

I recall that I had some cases which caused me to add the || buffer.isAutoCommit() to this condition, because otherwise certain newly created records in temp-tables which should have been flushed, were never being flushed. I don't recall at the moment what those cases were. However, I think when someone passes false for the flush parameter to the Validation c'tor, there is a reasonable expectation that the record will be validated only, not

flushed. So, the code above is not quite right. Certainly, when one arrives here through the validation-only code path you have shown, the record should not be flushed, even if the buffer is in an auto-commit state.

So, that's likely the problematic code, but I don't have a solution in mind, which does not break some other cases that this change got working...

You re-enabling the unique constraints seems to fix the accumulation of "validated" records by disallowing them, but (assuming this is where the problem is occurring) that is only by side effect. We should never be trying to flush them in the first place, in your scenario.

> I think this kind of code is used in a few other places. What do you mean by "_record validation in the Validation class is done by query_"?

Temp-table records are always validated by this code (also in Validation.validateUniqueIndices(BitSet)):

```
        if (multiplex != null || !flush)
        {
           // validate by executing unique index queries
           validateUniqueByQuery(check);
        }
```

That is, we don't rely on SQL unique constraints (since there are none), and always execute the validateUniqueByQuery method for temp-tables.

What I don't fully understand at the moment is that this code executes before the flush() call above, so invalid records should be caught and rejected by validateUniqueByQuery before they ever get to the flush() call. Why would invalid records accumulate, and why would re-enabling the unique constraints fix the problem? Perhaps the answer lies in what the value of the BitSet parameter is at the time of the call. Maybe we are only validating a subset of the indices in your code path, when we should be using all of them?

#3 - 12/29/2020 05:00 PM - Ovidiu Maxiniuc

- Status changed from New to WIP

I continued with investigation of the issue. The validateMaybeFlush() uses BaseRecord.getUnvalidatedIndices(int) to check whether to call the validateUniqueIndices() (where the Validation.validateUniqueIndices(BitSet) is called).

The problem is getUnvalidatedIndices() returns an empty BitSet. This is correct because none of the field components were ever touched. There was customer which was just flushed to database. It had all fields set to the default values, including the single index-field, customer-num. As the second customer is processed, also with all fields untouched, the method does not see any index to be invalidated because it does not check the database content, only the status flags of the index fields. But these look pristine, as the record was not touched in any way. As result, the new customer is flushed assuming no collision will happen.

If the unique index was in place, the SQL would have reported that there are two customer s with same null customer-num.

Here is the isolated testcase:

```
DEFINE TEMP-TABLE customer
   FIELD customer-num AS INTEGER
   FIELD customer-name AS CHAR
   INDEX idx-num AS UNIQUE customer-num.

CREATE customer. customer.customer-name = "1".
CREATE customer. customer.customer-name = "2".
CREATE customer. customer.customer-name = "3".
CREATE customer. customer.customer-name = "4".

FOR EACH customer:
   MESSAGE customer.customer-num customer.customer-name.
END.
```

It will print:

```
0 1
0 2
0 3
0 4
```

All customer-num are 0, even if a unique index exists (in legacy code) for it.

#4 - 01/26/2021 04:40 PM - Eric Faulhaber

- Status changed from WIP to Review

- Assignee set to Eric Faulhaber

- % Done changed from 0 to 100

I have committed to 3821c/11953 a fix for the failing test case in #5056-3. Please review. I put it in 3821c because it modifies an area of the code that is core to the ORM/persistence implementation, and I want it to receive broad testing.

There is still a deviation in behavior, in that the 4GL reports the unique index violation three times before ending the procedure, whereas FWD only reports the first conflicting record and ends the procedure. I don't think this deviation is about the flushing behavior per se, but it is worth noting that something is not quite right about the block error handling in this case.

Ovidiu, please either temporarily port this fix to branch 4397a or rebase that branch to pick up the fix, then test with the original scenario(s) which prompted you to open this issue. From my point of view, this issue is fixed, but we can change status back if the original problem is not resolved.

#5 - 01/27/2021 01:53 PM - Ovidiu Maxiniuc

Just inspecting the code it looked fine. I rebased and run the testcases to be sure. I got different behaviour, but not the expected one. The problem is that the new validation only happens at buffer release, not at buffer validation. And I need this to happen when validation is invoked, in order to know whether the new record will fit into the table, and depending on the MERGE/REPLACE mode to act accordingly.

I created the following testcase:

```
DEFINE TEMP-TABLE customer
   FIELD customer-num AS INTEGER
   FIELD customer-name AS CHAR
   INDEX idx-num AS UNIQUE customer-num.

CREATE customer. customer.customer-name = "1".
VALIDATE customer NO-ERROR.
MESSAGE "Validated" customer.customer-name "?:" ERROR-STATUS:GET-NUMBER(1).

CREATE customer. customer.customer-name = "2".
VALIDATE customer NO-ERROR.
MESSAGE "Validated" customer.customer-name "?:" ERROR-STATUS:GET-NUMBER(1).

CREATE customer. customer.customer-name = "3".
VALIDATE customer NO-ERROR.
MESSAGE "Validated" customer.customer-name "?:" ERROR-STATUS:GET-NUMBER(1).

CREATE customer. customer.customer-name = "4".
VALIDATE customer NO-ERROR.
MESSAGE "Validated" customer.customer-name "?:" ERROR-STATUS:GET-NUMBER(1).

FOR EACH customer:
   MESSAGE customer.customer-num customer.customer-name.
END.
PAUSE MESSAGE "all done.".
```

In OE it prints:

```
Validated 1 ?: 0
Validated 2 ?: 132
** customer already exists with  0. (132)
** customer already exists with  0. (132)
** customer already exists with  0. (132)
```

I assumed the last 3 messages were generated because the VALIDATE statements have NO-ERROR option and when customer buffer is released the error is not handled otherwise. But since my subsequent MESSAGE s are not printed and neither the final DISPLAY output, it is clear that it is not. It's strange that the message is emitted thrice.

With your patch, FWD will only check the indexes at buffer release time (when creating customer "3").

```
Validated 1 ?: 0
Validated 2 ?: 0
** customer already exists with customer-num 0. (132)
```

I am trying now to adjust your patch to work for manual validation, too. I'll keep you posted.

#6 - 01/27/2021 02:29 PM - Eric Faulhaber

The patch was based on knowing that the Validation instance was invoked for the purpose of flushing, but your point makes sense. We may need to introduce an new full flag to the Validation class (meaning, do a full validation of all remaining, unvalidated indices), and a new parameter to the c'tor to set it. This will have to flow backward through the call path to this c'tor (RB.validateMaybeFlush and its callers). I can make the change, if you want.

#7 - 01/27/2021 02:48 PM - Ovidiu Maxiniuc

Yes, please add the change to 3821c. I inspected the code (the four places where validateMaybeFlush() is called) and it looks to me that all the cases require the flag passed as same value, true.
Thank you.

#8 - 01/27/2021 03:15 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> Yes, please add the change to 3821c. I inspected the code (the four places where validateMaybeFlush() is called) and it looks to me that all the cases require the flag passed as same value, true.

I think that's too aggressive. For the moment, I am only passing true for the call paths from RB.validate(Record) and RB.flush(boolean). I am not 100% sure about RB.endBatch(boolean). Maybe this needs it, too. But, RB$Handler.invoke must pass false, because validation for a new record (when not in auto-commit mode) must be deferred until the sooner of (a) all elements of an index are updated, or (b) some other event forces it.

#9 - 01/27/2021 03:37 PM - Ovidiu Maxiniuc

Does it make sense to check whether the buffer is Temporary ? Because the permanent tables have all indexes in place and presumably working. Only the TEMP-TABLEs are stripped of their unique indexes.

#10 - 01/27/2021 03:50 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> Does it make sense to check whether the buffer is Temporary ? Because the permanent tables have all indexes in place and presumably working. Only the TEMP-TABLEs are stripped of their unique indexes.

Where do you mean? We have tests within Validation that differentiate logic based on whether a record is from a temp-table.

I believe I have fixed the problem at the Validation level, but we have a higher level problem that is preventing the validation error from being reported to/by ErrorManager. The VALIDATE statements are converted to BufferImpl.validate() calls. That method is implemented like this:

```
try
{
   buffer().validate(false);
}
catch (ValidationException e)
{
   // should never happen
}
```

As you can see, this passes false to RB.validate(boolean). The false parameter indicates not to throw a validation exception. So, even though ValidationException is being thrown by the Validation object, it is caught by RB.validate(boolean) and is eaten, rather than being reported to ErrorManager. RB.validate(boolean) returns false, but BufferImpl.validate() ignores the return code. Any idea why it was implemented this way?

#11 - 01/27/2021 04:15 PM - Eric Faulhaber

BTW, with my local changes, when I change the call from BufferImpl.validate to pass true to RB.validate(boolean), I get something which appears closer to the correct behavior, but the error handling is still not right:

```
Validated 1 ?: 0
Validated 2 ?: 132
Validated 3 ?: 132
Validated 4 ?: 132
0 1
all done.
```

It seems the explicit calls to the VALIDATE statement are maybe incorrectly marking the unvalidated state we track for the record as validated, even though validation failed. The implicit flushes of the records should trigger re-validation and produce errors, preventing the records from being flushed.

In any case, it seems we need to fix BufferImpl.validate() (and probably RB.validate(boolean)), because it seems the former method will never report an error the way it's currently written, no matter what happens with validation. Any input on the thinking behind the current implementation would help to direct a fix.

#12 - 01/27/2021 04:16 PM - Ovidiu Maxiniuc

Yes, the method implementation does not make sense. The caller will get nothing in return. Nice catch! I think it should be rewritten to something like:

```
      try
      {
         buffer().validate(true);
      }
      catch (ValidationException e)
      {
         // rethrow as 4GL error condition
         ErrorManager.recordOrThrowError(e.getNumber(), e.getMessage(), false, false, false);
      }
```

Which is pretty close to the implementation of for BUFFER-VALIDATE() method, found below.

However, from FILL method (BufferImpl.fill()) I use:

```
   validated = buffer().validate(false);
   if (!validated)
      [...]
```

It does not work here, too.

#13 - 01/27/2021 04:42 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> Yes, the method implementation does not make sense. The caller will get nothing in return. Nice catch! I think it should be rewritten to something like:
> [...]
>
> Which is pretty close to the implementation of for BUFFER-VALIDATE() method, found below.

The part you propose inside the catch block seems redundant with the logic in RB.reportValidationException, which is called when the throwValidationException parameter is true. This could result in the error being reported to ErrorManager more than once for the same error, which I think is incorrect. Can't we let RB.reportValidationException do its thing and just catch and eat the ValidationException here, knowing it already has been processed? In silent error mode, this will result in us catching and eating the ValidationException (which will have been logged by the ErrorManager already). In non-silent mode, ErrorConditionException will be thrown below, and will not be caught by BufferImpl.validate(), instead propagating up the stack until handled by BlockManager.

#14 - 01/27/2021 05:04 PM - Ovidiu Maxiniuc

Actually my first idea for validate() was as simple as:

```
buffer().validate(true);
```

But this does not work since the called validate(boolean) might throw an ValidationException which must be handled. So I caught it and do the single logical thing to do. The fact that the code is similar to final part of RB.reportValidationException() does not mean they will double. Once one of then execute will exclude the other since both of them transform a ValidationException to an ErrorConditionException.

#15 - 02/18/2021 05:10 PM - Eric Faulhaber

- File rbval.patch added

Ovidiu, please test whether the attached patch resolves your original issue(s). It gets both of the test cases you provided above pretty close to the same behavior. One of the ** customer already exists with customer-num 0. (132) messages is still missing in each case, but I believe the validation logic is closer to what it should be.

Basically, I changed the code to make sure that explicit DMO validation does not change the DMO state (i.e., which indexed fields have been touched and the overall validity state of the DMO). Only implicit/organic validation caused by DMO updates now updates the state.

If your results show improvement, I will commit the patch to 3821c.

#16 - 02/18/2021 06:19 PM - Ovidiu Maxiniuc

Eric,
I am sorry to inform you that the patch is not making the testsets behave better. In fact, the test stops much earlier than expected. So it is not yet ready to be committed.
I had a look at the patch, but I have not debugged it yet. I'll be back with more input later, tonight or tomorrow.

#17 - 02/18/2021 06:27 PM - Ovidiu Maxiniuc

Eric,
I looked deeper into the server's log (there are a lot of java.lang.IllegalStateException: Empty scopes in UniqueTracker in there) and found the cause for the premature end of the testsets: the MathOps: I need to update some FWD code and reconvert the testset. The porting from integer to int64 of some API does a real mess. So do not put any effort in this issue yet. I am not sure I will manage to do this tonight, but I will do it first thing in the morning.

#18 - 02/19/2021 07:46 PM - Ovidiu Maxiniuc

I used your patch to run the xfer tests and things have changed. It is not perfect, but I cannot tell exactly why.

I isolated a testcase where a dataset is copied over another using REPLACE mode. For each table of the destination, the primary unique index (on customerNum) is used to determine if a record exists for each of the source record. I checked the records and they exist in both tables (in fact there are 3 key duplicates which whould be replaced), so the _find in TemporaryBuffer:3478 should return true. But it does not. The query predicate (in this particular case WHERE ttOrderCustomer.customerNum=ttCustomer.customerNum) is, yet, correct. As result, the existing record is ignored and a new record is created for the same index key. Of course, it will fail validation so ** ttOrderCustomer already exists with customerNum 1. (132) is printed and the test will fail, eventually.

I will continue my investigations and report back.

#19 - 02/22/2021 07:43 PM - Ovidiu Maxiniuc

I found the cause for the 'regression' of your patch. It is probably not a single thing, but a couple of issues which went unnoticed until now.

I started my debugging and the first thing that I discovered was that the FastFindCache was returning NO_RECORD objects when my buffer().sqlTableContent(0) clearly showed the table content. Evidently there was an cache issue: the cache was not invalidated when the record was created and persisted.

I tracked back where those records were created and I found the main cause. In cases when multiple records were created at once (like copying tables) I used tempBuf.instantiateDMO() to create new instances of needed TempRecord type, avoiding this way the heavy-weight of calling RecordBuffer.create() (no locking, no triggers, no dirty checks). Also this allowed to manage locally the links to eventual before-table, or the before-records themselves to be created. The problem is, doing so the FFC is also avoided: the new record is not loaded into the buffer and neither the FFC manually invalidated. As results, the FFC was not aware of the newly created records, thus giving invalid results to queries which, by chance, were also internal implementation of other table copy methods.

I am reviewing now usages of instantiateDMO() methods for several places where multiple records are created simultaneously (copy tables, get changes) where it is used directly, as a performance leverage, to decide whether the create() method is appropriate to be used or other additional code need to be added.

#20 - 02/23/2021 06:34 PM - Ovidiu Maxiniuc

Similar to the patch proposed in #4884-15, I have fixed the cases identified above. The three main xfer testcases are cleaner, but they still end up prematurely. It seems that the patch is good, but before committing, I would like to investigate the cause for the other new crashes. They occur in read xml/json most likely when index collisions occur between the existing data in tables and the records deserialized.

#21 - 02/24/2021 09:25 PM - Ovidiu Maxiniuc

Eric Faulhaber wrote:

> Ovidiu, please test whether the attached patch resolves your original issue(s). It gets both of the test cases you provided above pretty close to the same behavior. One of the ** customer already exists with customer-num 0. (132) messages is still missing in each case, but I believe the validation logic is closer to what it should be.
>
> If your results show improvement, I will commit the patch to 3821c.

Yes. I can say for certain that it is, for certain, a nice improvement. Thank you! I was seeing some failing tests but they were surfaced issues on my side. Please commit the patch. I will also commit my changes after that.

#22 - 02/25/2021 01:21 PM - Eric Faulhaber

- Status changed from Review to Test

Patch committed as 3821c/12065.

#23 - 02/26/2021 02:11 AM - Eric Faulhaber

Can this issue be closed?

#24 - 03/12/2021 04:00 PM - Ovidiu Maxiniuc

The validation works correctly for temp-tables now. It does not really make sense why OE shows error 132 multiple times for same event.
The answer is Yes.

#25 - 03/12/2021 04:17 PM - Eric Faulhaber

- Status changed from Test to Closed

Files

rbval.patch	12.5 KB	02/18/2021	Eric Faulhaber