Database - Bug #7421

Check only the indexes that were changed when using Validation.checkMaxIndexSize

06/08/2023 08:42 AM - Alexandru Lungu

Status:	WIP	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Dănuț Filimon	**% Done:**	80%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD	**version:**	

Description

History

#1 - 06/08/2023 08:46 AM - Alexandru Lungu

Validation.checkMaxIndexSize is checking all indexes to detect if the maximum size was exceeded. This happens only on validate after an insert/update. However, not all indexes should be checked, but only those that have an updated field and are candidate to exceeding the maximum size.

There is a comment in Validation:

```
// TODO: this check needs to be split apart to only check those indices needing it, like we do with
// unique constraint validation
```

For unique indexes we do this already because checking one unique index is time consuming. Apparently, I have a POC with 78,407 calls to Validation.checkMaxIndexSize that have an own time of 252ms. I hope this can be severely reduced.

#2 - 06/08/2023 11:09 AM - Eric Faulhaber

Good idea. BaseRecord.getDirtyIndices already is invoked in several places in Validation. Can we re-use the values returned? BaseRecord.getDirtyIndices may itself be expensive (this is my educated guess, though it is not confirmed that this is its own bottleneck). So I don't want duplicate invocations of this, if it can be avoided easily.

I just noticed that neither MariaDbLenientDialect nor MariaDbDialect override the implementation of Dialect.getIndexLengthLimit(). The latter returns 0, which is not appropriate for MariaDb, so we should override this method in MariaDbLenientDialect. I don't recall the exact limit at the moment.

Also, note that for applications which don't care about enforcing the legacy, Progress index length limit, we can set persistence/max-index-size to 0 in the directory (this probably should be renamed to persistence/legacy-max-index-size). Otherwise, we do this check by default, even if the database dialect doesn't need it. The flip-side of this is that we will do the check if the dialect needs it, even if the legacy check is disabled.

#3 - 06/13/2023 07:49 AM - Alexandru Lungu

- Status changed from New to WIP

- Assignee set to Dănuț Filimon

#4 - 06/15/2023 05:18 AM - Dănuț Filimon

Eric Faulhaber wrote:

> BaseRecord.getDirtyIndices already is invoked in several places in Validation. Can we re-use the values returned?

From the creation of the Validation instance until the call to Validation.checkMaxIndexSize, BaseRecord.getDirtyIndices is not called at all. This method is not what we need here, I've noticed that indexed properties of indexes created on multiple fields are ignored, while we require them to check the index size.
For example: If you have a unique index on two fields, where each field is set, getDirtyIndices will not find the index when updating the first field, but knows it is part of one. The second field is found with no problems.

I've noticed that providing an offset that is **greater or equal than 0** results in only one index being changed, this means we can iterate the unique/non unique indices to find the index that needs to be checked. There is another case where multiple (not all) indexes can be updated when using ASSIGN, in this case the program will continue using the previous implementation. I think this case can also be improved using a slightly modified getDirtyIndices that doesn't ignore index fields (example above), but it might be an expensive call if all indexes are changed so I need a second opinion on this.

These changes for checking for a single index are **committed to 7421a.rev14626**, I will continue with testing and profiling.

#5 - 06/15/2023 07:48 AM - Dănuț Filimon

I created a test using a temporary table with 50 fields, all being part of unique/non unique indexes. It consisted of creating 25k records and the results obtained using VisualVM were:

Method	Total time Before (ms)	Total time After (ms)	Difference (%)	Invocation count
Validation.validateMaybeFlush	103245	69182	-32.992%	1250050
Validation.checkMaxIndexSize	64344	29431	-54.259%	1250050
BaseRecord.getDirtyFieldIndex	-	915	-	-/2500100

#6 - 06/30/2023 05:50 AM - Alexandru Lungu

- Status changed from WIP to Review

- % Done changed from 0 to 100

> I've noticed that providing an offset that is greater or equal than 0 results in only one index being changed, this means we can iterate the unique/non unique indices to find the index that needs to be checked. There is another case where multiple (not all) indexes can be updated when using ASSIGN, in this case the program will continue using the previous implementation. I think this case can also be improved using a slightly modified getDirtyIndices that doesn't ignore index fields (example above), but it might be an expensive call if all indexes are changed so I need a second opinion on this.

I will check after we get 7421a reviewed/profiled. I am not aware how many checkMaxIndexSize are hit from single updates vs ASSIGN. You can compute a statistic on this. So will I.

> These changes for checking for a single index.

Danut, what do you mean by single index. I read your statement, but I don't understand what you mean by "single-index". If you update a field (i.e. tt.f1 = 2), there may be multiple indexes that needs reconstruction, right? Did you mean "single-field update" (which is the opposite of multiple field update provided by ASSIGN)?

I see several javadoc and method names implying "single index". Do you mean strictly addressing create index on tt (f1)? There are very few such cases, so I think this implementation is very limited.
Also, I reviewed your code and noticed uIndexes = dmo.getDirtyFieldIndex(offset, true);. This is confusing uIndexes implies multiple indexes, while getDirtyFieldIndex implies a single index.

To get this straight, if I run tt.f1 = 2, where I have 3 indexes on (f1), (f2) and (f1, f2), will these be:

- checked for max-size, but with the old implementation going through all 3 indexes
- checked for max-size, but with the new implementation for both (f1, f2) and (f1). **This is the expectancy from this task**
- checked for max-size, but with the new implementation only for (f1). What happens to (f1, f2)?
- nothing is checked for max-size

Danut, the implementation needs a bit more consistency in javadoc / method names / comments to better get a grip of what is going on.

#7 - 07/10/2023 04:57 AM - Dănuț Filimon

Alexandru Lungu wrote:

> Danut, what do you mean by single index. I read your statement, but I don't understand what you mean by "single-index". If you update a field (i.e. tt.f1 = 2), there may be multiple indexes that needs reconstruction, right? Did you mean "single-field update" (which is the opposite of multiple field update provided by ASSIGN)?

You are right, there may be multiple indexes that needs to be reconstructed. The current implementation will only reconstruct a single index (first found) which is wrong.

I **committed 7421a/rev.14627** which fixes this mistake. I renamed getDirtyFieldIndex to getDirtyFieldIndices which will return all indices of a field that is updated. Now if you have 3 indices on (f1), (f2) and (f1, f2) and you update f1, the indices that will be returned are (f1) and (f1, f2).

I will compute a statistic for single-field update/ASSIGN and post an update soon.

#8 - 07/10/2023 06:42 AM - Dănuț Filimon

Dănuț Filimon wrote:

> I committed 7421a/rev.14627 which fixes this mistake.

7421a was rebased with trunk/rev.14649 which brings this commit to **7421a/rev.14651**.

#9 - 07/10/2023 08:55 AM - Dănuț Filimon

I tested a large customer application for ~45 minutes and managed to extract a total of **907392** statements, here's an overview of the data:

Statement	Count	% from Total	Total skipped	% from Count (1)	Only unique	% from Count (2)	Only nonunique	% from Count (3)	Both	% from Count (4)
CREATE	702329	77.4008	684055	97.398	8337	1.187	9910	1.411	27	0.0038
ASSIGN	205063	22.5991	39936	19.474	38488	18.768	48661	23.729	77978	38.026

Around **80%** of the statements are skipped which means a lot less time is spent reconstructing the indices. I actually didn't expect any assign statements to be skipped, but I should've thought that there might be tables that don't have any indices defined.

#10 - 07/12/2023 07:20 AM - Alexandru Lungu

Review of 7421a

- I am OK with the changes. Well done!
- The only point left here is in BaseRecord.getDirtyFieldIndices. You don't actually need filter. You shall only iterate through the indices array. Note that filter is never updated and is always full of 1. So isEmpty is true only when indices.length is 0. Also, int i = filter.nextSetBit(0); i >= 0; i = filter.nextSetBit(i + 1) can be rewritten into int i = indices.length - 1; i >= 0; i--.
- Compress the array return using return dirty == null ? null : dirty.toArray(new BitSet[dirty.size()]);
- if dirtyOffset < 0, you can simply return indices. Do this short-circuit. If you use such short-circuit, you can remove dirtyOffset >= 0 check for good.
- if the dirtyOffset isn't indexed (use recordMeta.getAllIndexedProperties()), you can return null straight-away (without iterating all indexes). **Please double-check this hypothesis; especially that dirtyOffset position matches the position in the bitset**!

#11 - 07/13/2023 03:27 AM - Dănuț Filimon

Committed 7421a/rev.14652. Added the changes mentioned in #7421-10.
Alexandru Lungu wrote:

> Review of 7421a
>
> - ...
> - if the dirtyOffset isn't indexed (use recordMeta.getAllIndexedProperties()), you can return null straight-away (without iterating all indexes). **Please double-check this hypothesis; especially that dirtyOffset position matches the position in the bitset!**

I did a few checks with extents and the dirtyOffset matched the correct position in the bitset each time. I also noticed that the properties in the bitset are ordered as follows: properties without an index, indexed properties and properties with extent.

#12 - 07/18/2023 10:39 AM - Alexandru Lungu

I am OK with the changes in 7421a/rev.14652.

Planning to test and profile them asap.

#13 - 07/19/2023 11:33 AM - Alexandru Lungu

- Status changed from Review to WIP

- % Done changed from 100 to 50

Good news: Testing shows no regression. Profiling shows some improvement, currently -0.8%. However, I am not quite stable with my profiling environment, so this might be slightly inaccurate.

I've seen multiple cases where indices is an empty BitSet. You can short-circuit this case: simply return null. Otherwise, the code will check if the current field is indexed - but this is not needed, because either way null will be returned. This might not be needed after you consider what is written

below. Also, dirtyOffset >= 0 is redundant. Finally, please remove the TODO from validateMaybeFlush.

However, the cause of the "modest improvement" is due to the following statistic (for getDirtyFieldIndices) I have on another customer application:

Scenario	Unique	Non-unique	Expected optimization
negative offset	28.289	28.289	no optimization done by #7421 - we do this check twice (for unique and non-unique)
field not indexed	43.949	43.949	these are skipped now - we do this check twice (for unique and non-unique)
empty array returned / null	2.761	3.038	such indexes will no longer be checked
array of size 1 is returned	3.122	2.844	only these indexes will be checked
array of size > 1	0	1 (array of size 3)	not enough to be relevant

For the "negative offset" scenario, we shall at least check if one of the dirty properties is indexed or not. I see some scenarios of ASSIGN cases over fields which are not indexed. I think we can rule-out index size checking if there is no update over indexed fields.

- You can use the current getDirtyIndices just to check if there is any index that is dirty. If this returns an empty bit-set, you can simply return. Note that this also covers the dirtyOffset case, so you can remove the indexedProps check from getDirtyFieldIndices, if you already use getDirtyIndices.
- **However, you should fine tune this solution, so please take a look on BaseRecord.getUnvalidatedIndices**. This will retrieve a bit-set of unique indexes that are invalidated. The same can be used to retrieve a bit-set of non-unique indexes that are invalidated. In that case, you can replace unvalidated with dirtyProps. The code index.intersects(dirtyProps) is really nice as it can check if the index has a dirty prop.
- **Lastly, the whole code in checkMaxIndexSize looks conceptually wrong to me: iterate all indexed fields and for each field iterate all indexes so the size is incremented little-by-little. There is clearly a better solution to iterate all indexes, and for each index iterate all its fields. This way, we can intersect dirtyProps with the index fields and skip indexes that don't require size checking. For those that need size checking, we just do that.**

I will do another profiling after this is completely optimized. **Danut, please prepare of a full refactoring of checkMaxIndexSize.** I don't think "staying on the surface" with getDirtyFieldIndices is enough; we need deeper tested changes. Don't just precompute the indexes we need to iterate; go index by index and skip the ones that don't intersect dirtyProps. Note that you can always short-circuit the process for: (dirtyProps.isEmpty() || (dirtyOffset >= 0 && !allIndexedProps.get(dirtyOffset)).

Mark #7421 and #7496 your top priority.

#14 - 07/19/2023 02:56 PM - Eric Faulhaber

AFAIR, we are always doing the "legacy" max index size check (i.e., the one that enforces a Progress database limit), even when the replacement database has no index size limit. Is that still the case? This seems to be unnecessary effort in most cases. I would suggest we turn this off by default and enable it to be honored through a configuration option, if for some reason, an application wants to abide by this limit.

#15 - 07/20/2023 02:51 AM - Alexandru Lungu

- % Done changed from 50 to 80

> I will do another profiling after this is completely optimized. Danut, please prepare of a full refactoring of checkMaxIndexSize. I don't think "staying on the surface" with getDirtyFieldIndices is enough; we need deeper tested changes. Don't just precompute the indexes we need to iterate; go index by index and skip the ones that don't intersect dirtyProps. Note that you can always short-circuit the process for: (dirtyProps.isEmpty() || (dirtyOffset >= 0 && !allIndexedProps.get(dirtyOffset)).

This is wrong on my side! Currently, there are NR_INDEXED_FIELDS * INDEX_COUNT iterations. According to my suggestion, the complexity will get close to NR_FIELDS * INDEX_COUNT, which is way larger. Danut, please dismiss my suggestion. At best, we can skip the indexed fields if their flag in dirtyProps is set - but in this case we need to make sure that **all** fields of the index are non-dirty. This is too much effort for too little and we may introduce overhead.

I think we can leave the implementation as it is now (only address #7421-13 short-circuits). Basically, we will skip the ASSIGN cases from this optimization, which is fine to me, as there is no clear fast solution for them.

> AFAIR, we are always doing the "legacy" max index size check (i.e., the one that enforces a Progress database limit), even when the replacement database has no index size limit. Is that still the case? This seems to be unnecessary effort in most cases. I would suggest we turn this off by default and enable it to be honored through a configuration option, if for some reason, an application wants to abide by this limit.

Well, we have an option for that persistence/max-index-size, but it is set across all databases. If it is not set, it will default to the 4GL limit of 1971. Note that there is some specific size computations to 4GL which makes a distinction between unique and non-unique indexes, so for that matter, the size of an index in 4GL may not be the same as the size of an index in other database we use.

I think we can leave the implementation in #7421 as it is now as it optimizes the size check anyway (for legacy mode), with no regression. **Danut, please check if PG, MariaDB and H2 are enforcing an index size limit**. We shall know if we can ever encounter such index size issues. For the customer I am doing my tests, we can set max-index-size on 0 (no check), but I must be sure that PG or MariaDB at least are not enforcing limits.

#16 - 07/20/2023 05:21 AM - Dănuț Filimon

Committed 7421a/rev.14653 Short-circuited indices when it is empty and removed redundant check on dirtyOffset. I also moved the getDirtyFieldIndices public method since it was placed between private methods.

Alexandru Lungu wrote:

> **Danut, please check if PG, MariaDB and H2 are enforcing an index size limit**. We shall know if we can ever encounter such index size issues.

MariaDB enforces a limit of **3072** bytes.
PostgreSQL enforces a limit of **8191** bytes.
H2 does not enforce a limit.

I noticed that PostgreSQL enforces the limit when the values are used (e.g. inserting a record), while MariaDB enforces it when the index is created. MariaDB also doesn't allow an index to be created with columns that do not specify a length.

#17 - 07/20/2023 05:59 AM - Alexandru Lungu

At least for FWD-H2 we can skip the "check index size". In 4GL, temp-tables are subject to index size limitation, so here persistence/max-index-size actually makes sense.
For PostgreSQL and MariaDB is quite troublesome. If we use persistence/max-index-size, we may end up with SQLException later on, which is not quite good IMHO.

#18 - 07/20/2023 03:13 PM - Ovidiu Maxiniuc

Dănuț Filimon wrote:

> MariaDB enforces a limit of **3072** bytes.
> PostgreSQL enforces a limit of **8191** bytes.
> H2 does not enforce a limit.

There is getIndexLengthLimit() method in Dialect class which should give these responses at runtime. It seems that:

- we forgot to implement it in MariaDB;
- disable the check by returning 0 for PostgreSQL;
- the most constraints has the SQL Server 2012, only 900 bytes, which is missing from your list.

LE: There are two kind of enforcements here:

- statically (or syntactic), which happens when the schema is declared (index is created). The database know the (maximum) dimension declared for a field and will reject creation of the index if the sum of column sizes passes the 3072 limit. In this cases, we do not need to check the index size for each record. MariaDB does this;
- dynamically (or semantically). The size of the index key is computed at runtime, for each record. In this case the sum of all maximum sizes of the index components may be larger than the accepted index size, but if the sum of the column sizes is less than the accepted size, then the record is accepted. 4GL is an example and PostGreSQL also.

Note that 4GL has variable size even for data types which have fixed known static sizes. For example, an int64 may take 1 to 9 bytes, depending on the actual content (see int64.getSize() method).

#19 - 07/21/2023 04:37 AM - Alexandru Lungu

> Note that 4GL has variable size even for data types which have fixed known static sizes. For example, an int64 may take 1 to 9 bytes, depending on the actual content (see int64.getSize() method).

checkMaxIndexSize is handling both SQL and 4GL limitations:

- 4GL limitation can be disabled, but it won't increase performance as long as the SQL limitation should be checked as well. Setting persistence/max-index-size on 0 is not quite a free lunch.
- The SQL limitation can't be disabled, so we will face them anyway (at least for some database engines). For H2 and MariaDB we can disable max-index-size to skip the index size check anyway. For PG and SQL Server, we should enforce the dynamic checking.

What about making persistence/max-index-size more flexible? Rename it to check-index-size and add:

- legacy mode: always check that the legacy value is honored. Default value now is true. **Eric, can we switch the default to false?**
- replacement mode: always check that the replacement database constraints are honored, if exists (PG and SQL Server). This is by default true. One can disable it risking to face PersistenceException.
- per-database: make these settings "overridable" per-database, including temporary database.
- global: make these

I don't think it makes sense to override the LEGACY max-index-size, right. So as you said, we can drop max-index-size and add check-legacy-index-size. There is no value is setting a specific upper-bound.

First optimization: reduce the number of engines / databases for which the check is done. If we make the legacy check default to false, H2 and MariaDB will benefit from a free performance boost.
Second optimization: reduce the number of indexes checked. This is already done in 7421a.

#20 - 07/21/2023 04:47 AM - Alexandru Lungu

> I just noticed that neither MariaDbLenientDialect nor MariaDbDialect override the implementation of Dialect.getIndexLengthLimit(). The latter returns 0, which is not appropriate for MariaDb, so we should override this method in MariaDbLenientDialect. I don't recall the exact limit at the moment.

> we forgot to implement it in MariaDB

Why should we? getIndexLengthLimit is in fact 3072, but will this help us at run-time in any way? If we already defined the schema and passed validation, why should we store the maximum index size of MariaDB and still do the checking? If we attempt to break the index size, we will have "Data doesn't fit in type"; MariaDB won't ever do "Index size exceeded". So the problem should be / is already handled somewhere else. Otherwise,

we may want to have a flag for getIndexLengthLimit, named static: true for the constraints at schema definition / false for the constraints used at run-time. checkMaxIndexSize will ever use the flag set on false.

#21 - 07/21/2023 08:45 PM - Ovidiu Maxiniuc

Alexandru Lungu wrote:

> checkMaxIndexSize is handling both SQL and 4GL limitations:

I know, I *might* have written some code which you are now trying to optimize :).

- 4GL limitation can be disabled, but it won't increase performance as long as the SQL limitation should be checked as well. Setting persistence/max-index-size on 0 is not quite a free lunch.
- The SQL limitation can't be disabled, so we will face them anyway (at least for some database engines). For H2 and MariaDB we can disable max-index-size to skip the index size check anyway. For PG and SQL Server, we should enforce the dynamic checking.

I agree.

> I don't think it makes sense to override the LEGACY max-index-size, right. So as you said, we can drop max-index-size and add check-legacy-index-size. There is no value is setting a specific upper-bound.
>
> **First optimization**: reduce the number of engines / databases for which the check is done. If we make the legacy check default to false, H2 and MariaDB will benefit from a free performance boost.

I am not sure how H2 works, but in case of MariaDB, the index limit is 3KiB. If we want to honour 4GL legacy limits (171 (old) and 1971) we cannot rely on SQL to fail validation for us. Since the limit of dynamic validation of OE is less than MariaDB's static index validation there are cases where a record will fit FWD's SQL but in legacy mode won't. These cases should be reported if we aim for full compatibility.

> **Second optimization**: reduce the number of indexes checked. This is already done in 7421a.

Clearly, I did not think about this at the moment when I implemented checkMaxIndexSize() validation as part of #4011.

> we forgot to implement it in MariaDB

> Why should we? getIndexLengthLimit is in fact 3072, but will this help us at run-time in any way? If we already defined the schema and passed validation, why should we store the maximum index size of MariaDB and still do the checking? If we attempt to break the index size, we will have "Data doesn't fit in type"; MariaDB won't ever do "Index size exceeded". So the problem should be / is already handled somewhere else. Otherwise, we may want to have a flag for getIndexLengthLimit, named static: true for the constraints at schema definition / false for the constraints used at run-time. checkMaxIndexSize will ever use the flag set on false.

For completeness. Maybe that API will be used with other utilities, for example for validating or adjusting schema based on exported data before setting it into database.

#22 - 07/26/2023 03:29 AM - Alexandru Lungu

I will redo the testing with **7421a/rev.14653** and max-index-size set on 0. This will reduce the number of indexes checks and will avoid the index-check for FWD-H2 completely.
3072 should be set for MariaDB, but right now my tests don't use MariaDB.