Database - Bug #6837

prevent unnecessary upper/rtrim injection in queries if the data type doesn't require it

10/12/2022 03:52 PM - Eric Faulhaber

Status:	WIP	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD		

Description

Related issues:

Related to FWD - Feature #2949: Alternative approach to handling case insensi...	**New**
Related to Database - Bug #7108: Simulate upper/rtrim directly in H2 using ca...	**Closed**

History

#1 - 10/12/2022 04:02 PM - Eric Faulhaber

In some cases, we are using upper and/or rtrim in WHERE and ORDER BY clauses to adjust textual columns that don't need it, due to the nature of the data type. Currently, this occurs in "lenient" mode in the MariaDB dialect for varchar columns, but it would also occur if we switched from text to citext in PostgreSQL, which is something I'd like to do.

The insertion of upper occurs at least during conversion, possibly at runtime. The insertion of rtrim occurs at runtime in the HQLPreprocessor.

Using these functions improperly can result in bad query plans, since indices will not match the column references in the WHERE or ORDER BY clauses.

I think the appropriate place to determine whether these functions should be injected is during FQL -> SQL conversion, with consultation to the active Dialect subclass.

#2 - 10/12/2022 04:24 PM - Ovidiu Maxiniuc

Eric Faulhaber wrote:

> The insertion of upper occurs at least during conversion, possibly at runtime. The insertion of rtrim occurs at runtime in the HQLPreprocessor.

In some cases, the conversion may be aware of some additional information, like when comparing a char string with a case-sensitive variable. If we do not do this at conversion time whenthe information on both operands is available, at runtime this might be impossible (because the variable -- in this case -- will be stored in a list of substitution elements, possible lazily evaluated).

> I think the appropriate place to determine whether these functions should be injected is during FQL -> SQL conversion, with consultation to the active Dialect subclass.

There is more. We already have cases where the same field type (character in ABL) is converted to different SQL type. For example, a field is usually converted to unrestricted text in PSQL, but in MariaDb it may be varchar(n) or text, if conversion decides the former is too big. This information is NOT available in DMO annotation; it only contain metadata common to all dialects.

#4 - 10/24/2022 06:34 PM - Ovidiu Maxiniuc

The dropping of the upper() / rtrim() functions injected during conversion must be done anyway. Initially, it will be done for all characters fields when the new MariaDbLenientDialect is detected, which will also propagate to MariaDbDialect. Later, after adding some kind of registry/collection on which we store the SQL columns types for each 4GL field, the operation will be more accurate, affected only the fields which require this. Even if this would be theoretically possible, we will not store this kind of information in DMO annotation (at least for 2 reasons: it is dialect dependent; the DDL is generated at a later/unrelated time during conversion).

Evidently, his will work incorrectly/slow for text fields, but their count is far small than varchar() s, and this is only a transition period.

For the "right way" implementation, we eventually need to talk how to store the missing type information: ~~DMO annotation,~~ some XML, or a local h2 database. For a similar issue in H2, we come up with __i and __s prefixes. But this is not really applicable here.

#5 - 10/24/2022 07:23 PM - Eric Faulhaber

I agree that we don't want dialect-specific metadata in DMO annotations. OTOH, I think it makes sense to store that metadata in DmoMeta instances at runtime, so it is available when needed to process a query.

The source of that information could be JDBC metadata, collected at server startup. If that data collection is slow, we could keep server startup relatively fast (important for development/testing, if not production) by launching a separate thread per database to do this work.

In the event the metadata is not yet available the first time it is needed to process a query, it could be collected from the query processing code as well, lazily.

#6 - 10/24/2022 08:41 PM - Ovidiu Maxiniuc

- Status changed from New to WIP

Eric Faulhaber wrote:

> I agree that we don't want dialect-specific metadata in DMO annotations. OTOH, I think it makes sense to store that metadata in DmoMeta instances at runtime, so it is available when needed to process a query.

DmoMeta could be a solution, even if it was also designed to be dialect independent. This class is used not only with the primary database but also with the dirty one. However, each dialect will be interested in specific properties and ultimately, they should live together.

> The source of that information could be JDBC metadata, collected at server startup. If that data collection is slow, we could keep server startup relatively fast (important for development/testing, if not production) by launching a separate thread per database to do this work.
> In the event the metadata is not yet available the first time it is needed to process a query, it could be collected from the query processing code as well, lazily.

If we go this way, the request for metadata from SQL server could be dialect independent (I am thinking of MariaDb's describe <table> / show [full] columns from <table>) if they prove to be faster than JDBC's DatabaseMetaData APIs. More than that, this background thread will be started only for specific databases. And we need to implement proper synchronizations.

#7 - 10/31/2022 09:33 PM - Ovidiu Maxiniuc

I committed an intermediary solution. The occurrences of properties are no longer wrapped in upper() and/or rtrim() if the dialect declared that it supports string data-types similar to 4GL. Also, no wrapping occur when the sorting criteria as composed in the order-by clause.

This is an intermediary commit because the final version must be aware of the actual type of the columns as not all strings in MariaDb behave the same.

Committed revision 14317.

#8 - 11/01/2022 09:19 AM - Greg Shah

- Related to Feature #2949: Alternative approach to handling case insensitivity added

#9 - 11/01/2022 09:29 AM - Greg Shah

Please ensure that the solution is built such that all dialects (not just MariaDB) can leverage this same approach. We have other requests for this (e.g. see #2949 for the request from a customer that uses both PostgreSQL and SQLServer). We are potentially considering these changes very soon (see #6898).

#10 - 01/31/2023 01:59 AM - Eric Faulhaber

What is left to do on the MariaDB side for:

- lenient mode?
- strict mode?

#11 - 01/31/2023 03:59 PM - Ovidiu Maxiniuc

- Subject changed from prevent unnecessary upper/rtim injection in queries if the data type doesn't require it to prevent unnecessary upper/rtrim injection in queries if the data type doesn't require it

The current implementation injects the upper/rtrim into FQLs of the queries at conversion time because it assumes at one of the dialects the application will be run on will not have automatic handling of CI/trimming like 4GL.

Later, at runtime, based on the value returned isAutoRtrimAndCi of the current dialect, the upper/rtrim functions are dropped while the FQL is preprocessed. From this POV, both MariaDb dialects do not need the functions to be injected because they both use the case-insensitive varchar(N) which ignores right-padding spaces.

If, for other dialects a similar data type is used for same reason, it is enough to override/implement isAutoRtrimAndCi and return true. Of course, the DDL generator should also be modified so that the indices to be composed of the pure column, not an expression or a computed column.

#12 - 02/03/2023 10:49 AM - Eric Faulhaber

It sounds like this task can be closed, then, correct?

#13 - 02/03/2023 10:51 AM - Greg Shah

It seems like note 11 is suggesting some changes to encode the logic in the dialects and fix the DDL generation.

#14 - 02/03/2023 05:25 PM - Ovidiu Maxiniuc

The changes are necessary only if/when we decide to switch from case-sensitive character data types to case-insensitive. The changes are localized within the respective dialect implementation (override isAutoRtrimAndCi and generate proper column types in tables). There is nothing to do ATM.

Note:
I wanted to manually test the behaviour in PSQL and I tried creating a table having a citext column. All I get is:

```
ERROR:  type "citext" does not exist
```

I understand this is a new data type (PSQL 11 and newer). I am using psql (PostgreSQL) 14.2 (Ubuntu 14.2-1.pgdg20.04+1). Do I need to enable it somehow?

#15 - 02/03/2023 06:17 PM - Eric Faulhaber

I'm pretty sure citext existed as an extension before PostgreSQL 11. I thought it was one of the core data types after that, though. Did you try something like this?

```
CREATE EXTENSION IF NOT EXISTS citext WITH SCHEMA public;
```

If this is necessary in the most recent versions, I wonder what that means for cloud implementations like Aurora, in terms of what they do and do not allow.

#16 - 02/03/2023 06:53 PM - Ovidiu Maxiniuc

Thank you. I tried it now and it worked. I was able to create a table and execute a quick select statement over a couple of inserted records. So it's seems that we will not have to inject the upper() function.

However, unlike MariaDB's strings, the right padding spaces are taken into account so the rtrim() still needs to be injected. From my tests: 'XYz' = 'xYz' but 'ABc' <> 'aBc ' if they are stored in citext fields.

And the not so good news do do stop here. Wrapping the case-insensitive citext in rtrim() will bring us back to square 1. That because the result of rtrim(citext) = text. With the above example, rtrim('ABc') = 'ABc' and rtrim('aBc ') = 'aBc'. But 'ABc' <> 'aBc' since they are now text, not citext :(.

Unless this can be configured somewhere else. Or if there is another data type for this?

#17 - 02/08/2023 04:00 AM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> Unless this can be configured somewhere else. Or if there is another data type for this?

I don't think so. It sounds like citext may not meet our needs, then.

Another option we could look into is combining the use of citext with custom operators, which would perform comparisons of their operands after applying a right-trim operation (presumably "manually" since as you note, the rtrim builtin function will return a text type in cases when we want citext).

PostgreSQL allows custom operators to be defined, but...

- I don't know the details of the implementation; I have not tried this before;
- it is unclear whether this is an allowable customization for cloud implementations, like Aurora.

#18 - 02/09/2023 10:42 AM - Alexandru Lungu

- Related to Bug #7108: Simulate upper/rtrim directly in H2 using case specific columns added