Database - Feature #2273

Feature # 1656 (Closed): add Microsoft SQL Server support

address remaining limitations of SQL Server needed to support legacy 4GL behavior

03/29/2014 04:03 PM - Eric Faulhaber

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Ovidiu Maxiniuc	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	Cleanup and Stablization for Server Features		
billable:	No	**vendor_id:**	GCD

Description

Related issues:

Related to Database - Feature #2296: update record buffer validation to preve...	**Closed**
Related to Database - Feature #2312: match Progress collation in a SQL Server...	**WIP**

History

#1 - 03/29/2014 04:55 PM - Eric Faulhaber

Please review the entire history log of #2141. While in the end we were able to generate DDL for a SQL Server schema, converted from a complex 4GL schema, there was a core issue that we were unable to address satisfactorily at that time. Namely, converted indexes with combined content > 900 bytes cannot be directly accommodated in SQL Server. It appears the practical limit of 4GL indexes is 1970 bytes, so it is possible/likely that we will encounter records during data migration and during runtime use, which cannot be inserted into tables constrained by certain indexes (i.e., when such records contain data in indexed columns which for a given index exceeds a combined length of 900 bytes).

The issue is really only of practical importance when dealing with converted character (in 4GL terms) fields. It is common in Progress to define indexes containing multiple fields, since the sorting behavior in Progress is so dependent upon indexes. When those indexes contain only fields of fixed-length data types like integer, decimal, logical, etc., this generally is not a problem. The problem becomes more likely when one or more character fields are present in an index.

Because an exported 4GL schema definition does not specify any enforced length limits on character fields, we do not have enough information at conversion time to impose any limit on the corresponding SQL data type. So, we end up (currently) defaulting to a data type of varchar(8000) for any converted character field. This is not a particularly elegant solution, since it probably is uncommon that most such fields actually require anywhere near this maximum. On the other hand, in certain cases, this maximum may not be enough, because to our knowledge, the practical limit for such fields in Progress is actually higher (32K, as far as we know).

This issue is easily avoided in PostgreSQL because that implementation offers the text data type, which is of variable length and requires no maximum length specification. This data type is also allowed in indexes without length limits. Because this is not the case in SQL Server, we need a solution which allows us to create indexes which properly index character data, which are guaranteed to not violate the 900 byte limitation.

#2 - 03/29/2014 04:57 PM - Eric Faulhaber

One possible approach is to provide additional inputs into the conversion process to properly limit the maximum lengths of character fields, based on

actual use. These limits would be honored at conversion time to produce appropriate DDL, and would be enforced at runtime to prevent invalid entries from being stored in the database. The latter would have to be done in such a way that would not break compatibility with the expected, legacy behavior, perhaps as part of the normal, 4GL-like validation performed when saving data.

Some advantages of this approach are:

- more clarity in the converted schema, since length limits are clearly defined;
- confidence that records will not be dropped/disallowed, because it will be known up front that all combinations of fields in an index will fit in that index - SQL Server will warn us if any indexes are poorly defined, such that they have the potential to drop records.

Some drawbacks to this approach are:

- it requires great discipline and accuracy on the part of the customer in terms of defining the limits appropriately;
- it requires additional infrastructure at conversion to process and integrate the limits in order to produce the proper DDL;
- it requires additional infrastructure at runtime to validate/enforce the limits;
- it may change the behavior of the application, because the runtime validation could determine data to be invalid that would have been valid before, due to stricter, post-conversion length limitations;
- it simply may not work for certain data set and schema definition combinations (i.e., where index sizes fall into the range > 900 bytes but < 1970 bytes).

Because of the limitations and potential failure of this approach for certain situations, it may have merit in some cases, but it is not a general-purpose solution to address the core problem.

#3 - 03/29/2014 06:35 PM - Eric Faulhaber

Perhaps a more general-purpose approach is to use computed columns in indexes in the place of converted character fields. I don't have the implementation idea fully worked out, but the general idea is to use a computed column to compute a fixed-length, compact, sortable value from the text value in a converted character field, and index the compact value instead of the full, text value. The compact value would have to be of a length such that no index in the database could ever exceed the 900 byte limit.

Some advantages of this approach:

- it is a general-purpose solution which does not impose new limitations on the length of fields that did not exist before;
- it does not require additional inputs to the conversion process and avoids efforts like customer audits to determine actual data lengths; this reduces the potential for human error;
- confidence that records will not be dropped/disallowed, because it will be known up front that all combinations of fields in an index will fit in that index - it will be known up front that no index can ever exceed the 900 byte limit.

Some drawbacks to this approach:

- it is difficult to implement: if the computed column algorithm is limited to the current piece of data alone, it likely will be lossy; if it uses surrounding records to determine the sortable value relative to others, it will be more complicated to implement, requiring additional data access(es) to do its work;
- it may cause noticeable performance overhead (most likely during data insert, though reads should be relatively fast);
- the compact, indexed values may "cluster" or run out of sufficient gaps between one another, which will cause problems with unique indexes or require redistribution efforts affecting multiple (possibly many) records; again, this is difficult to implement and may cause performance and/or unusual locking issues.

This idea needs to be fleshed out and vetted in greater detail. Please comment on the pros and cons of this approach, and contribute any design/implementation ideas that come to mind. It is not an approach we prefer to take, but it may be necessary.

Ideas for other approaches are welcome as well.

#4 - 03/29/2014 06:53 PM - Eric Faulhaber

Some more thoughts on the challenges of the computed columns approach...

- If done instead of the hard-coded length approach, the absence of the known maximum lengths of columns makes it difficult to bias the algorithm to use a properly representative subset of the text value to generate the compact value.
- In a lossy algorithm, two different strings could generate the same compact value. This could incorrectly violate a uniqueness constraint. This gets more complicated if there are other columns in the index, which in combination with the incorrectly non-unique value of the compact value, may cause an unexpected and possibly incorrect impact on the overall uniqueness of the index entry.

#5 - 04/01/2014 04:00 PM - Ovidiu Maxiniuc

Indeed, this is a non-trivial issue. Here are some of my thoughts:

1st approach (user filtering of the input):

- manually scanning the existing records won't suffice as the values inserted into tables at runtime are by their nature unpredictable.
- the input would be useful from an application expert. Using the theoretical lengths of each character fields we can compute the maximum theoretical size for each index to see if any passes the 900 bytes limit.
- in the exported .df files there are MAX-WIDTH and SQL-WIDTH clauses for character fields. Could these be used for some expected size calculation of the indexes ?

2nd solution is similar to a hashing function. Our problem can be solved by a hashing function that reduces the space of strings by aproximativelly 1/2 ratio.

- the entropy of the current and expected values is most likely different for each column so different compressing procedures can theoretically be implemented
- some algorithms that will decrease byte-size of a string: RLE, Huffman codes, attempts to make use of the bits that are unused by ASCII encoding.
- if an index has many (non-char) components the space available for the char fields decrease rapidly so a better compression ratio might be needed.
- I would discourage the lossy implementations and maybe ban them completly in unique indexes

A third approach would be to mix both previos ones and obtain a better solution.

Some other issue is that varchar in SQL server can only store 8 bit ASCII content. I know that Progress string is not limited to one byte but uses 2 bytes per character in database so the nvarchar would be the correct datatype. This means that actual Progress limitation is 4K.

Also, looking into SQL server specifications I see that there is one more restriction of maximum 16 columns per index, that is the same to Progress. Roughly, this leaves room for 50 bytes per field in average if all index components are strings.

#6 - 04/01/2014 04:27 PM - Constantin Asofiei

Eric, considering that we already have a locking mechanism above the physical DB layer and this requires that the DB records can not be changed while the P2J server is referencing them, have you considered keeping the indexes which exceed 900 bytes in an external storage? I'm thinking keeping alongside the MSSql DB a H2 DB resposible solely of enforcing the unique indexes and maybe also for sorting the records via an index.

#7 - 04/02/2014 09:57 AM - Eric Faulhaber

Constantin, can you elaborate how this might work? How would we engage the external indexes to perform a query, for instance? How would the mechanics of a new insert work across databases?

There are two potential issues I see off the top of my head:

- This would require management of another database for each SQL Server instance, which may not be acceptable to the customer (to their customers, specifically).
- We don't know for sure that there won't be external inserts/updates/deletes to the database, when the P2J server is offline.

#8 - 04/02/2014 11:08 AM - Constantin Asofiei

Eric Faulhaber wrote:

> Constantin, can you elaborate how this might work? How would we engage the external indexes to perform a query, for instance?

For querying, I'm not sure yet how the additional DB can help easily, the main goal is to be 100% sure the unique constraint is enforced. Although keeping a combined hash of all the indexed columns in an additionally, computed column, and enforce the unique constraint on that column might help.

> How would the mechanics of a new insert work across databases?

I guess table locks would be involved here, when inserting via P2J server.

> There are two potential issues I see off the top of my head:
>
> - This would require management of another database for each SQL Server instance, which may not be acceptable to the customer (to their customers, specifically).

Yes, this is a concern.

> - We don't know for sure that there won't be external inserts/updates/deletes to the database, when the P2J server is offline.

This can be enforced via a DB-level trigger, but accessing the H2 DB will be expensive (as we can't call java code directly from the trigger).

Anyway, I think this might not be a feasible after all. What I had in mind was a solution to move past the MSSQL limitations related to indexes and transfer this resposibility to another DB server, the complex part being to make these two communicate efficiently and consistently.

#9 - 04/02/2014 12:43 PM - Vadim Nebogatov

What do you think about using embedded HashBytes() function like http://www.brentozar.com/archive/2013/05/indexing-wide-keys-in-sql-server/ ?

#10 - 04/02/2014 01:11 PM - Ovidiu Maxiniuc

Yes, that is the idea. The computed column (the hashing result) would fit into SQL restriction and the INCLUDE would grant the speed of query executions. However, there are two issues here:

- even if the probability is low, still there are chances for collisions that would made UNIQUE indexed fail.
- the SHA2_512 hashing function is not a monotone functor between the varchar(max) and bit ^{512}. This means, the order of the query result will be different than expected (the records will be enumerated using the hashing order instead of original alphabetical plain text).

#11 - 04/02/2014 01:28 PM - Eric Faulhaber

Some good news out of today's meeting with the customer: they have not taken advantage of the newer, 1972 byte index limit in Progress, so their indexes all currently are governed by the older, 188 byte limit. This means that all existing data should easily fit within the 900 byte SQL Server limit. While this does not give us a long-term solution for future customers which have expanded to the 1972 limit, it does significantly simplify the problem we need to solve now.

So, we do not have to address the general-purpose solution in note 3 at this time. We do, however, have to:

- enhance the schema hint infrastructure to allow explicit size limits to be applied to converted character fields -- this information will have to be provided by the customer in the form of hints;
- determine and mimic the error behavior of Progress if one enters a record which violates the maximum index size (though I think it is OK to make that limit dialect-specific and/or configurable -- so while Progress will only allow 188 bytes, we could allow up to 900 for SQL Server, or skip the check altogether for PostgreSQL).

Greg suggested two alternative approaches to defining hard limits on all text fields. One was to limit the field size only for fields which participate in an index. The other was to use an unlimited data type (pending research to determine if there is one in SQL Server which can be used in an index), and letting the limit-checking validation we add in the persistence layer enforce the overall index size limit (ignoring SQL Server warnings about the 900 byte limit at schema definition time). The customer expressed an interest in limiting at least some fields, however.

There also was discussion about whether to use ISO-8859-1 (the current approach, both for the customer's Progress databases and for the converted database) or Unicode. The latter would increase the effort.

Next steps are to:

- research the options we have in terms of unlimited length text data (including participation of these data types in indexes and performance considerations);
- write test cases which violate the Progress index size limit to determine error conditions we need to handle;
- enhance the schema hint infrastructure to allow length limits to be specified as hints and to have that information integrated into the schema conversion process.

Please post all findings and proposed schema hint syntax in this issue.

Let's put the other ideas on hold for now, knowing that we may have to come back to them at a later date.

#12 - 04/02/2014 03:30 PM - Ovidiu Maxiniuc

- Status changed from New to WIP

Research the options we have in terms of unlimited length text data (including participation of these data types in indexes and performance considerations)

There are 3 main kind of character data in SQL:

- fixed size: char(n) and nchar(m), n < 8000, m < 4000
  can be key of indexes, if index total size > 900 will give an error
  can appear in INCLUDE clause of the index
- variable size: varchar(n) and nvarchar(m), n < 8000, m < 4000
  can be key of indexes, if index total size without them < 900 but with them > 900 a warning is issued
  can appear in INCLUDE clause of the index
- large data types: varchar(max) and nvarchar(max) also known in older SQL versions as text and ntext.
  cannot be in any part of indexes.
- sql_variant is some kind of polymorphic type by can have a maximum length of 8016 bytes.
  can also be part of an index as key
- xml - up to 2 GB of data (similar to text) holding well formed xml
  cannot be in key part of indexes but can appear in INCLUDE clause

The INCLUDE clause in the SQL index [[http://technet.microsoft.com/en-us/library/ms190806.aspx]] is particularly interesting from the performance point of view. The fields that cannot be in part of actual index because of some constraints can be stored within index tree (in leaves nodes) so when the index is used, their value is instantly available along with other key components of the index.

There is another limit in SQL Server. This maximum allowable table row size is 8060 bytes (including bytes of internal overhead). [[http://technet.microsoft.com/en-us/library/ms143432.aspx]]. I believe the Progress limit in this case is about 32K.
This is strictly enforced in SQL Server 2008 and for fixed character fields in SQL2012 (the error will be printed at table creation time).
It looks like Microsoft have done some work here, so in SQL 2012 this 8060 bytes limits can be broken by varchar and varchars. The row-overflow storage enables variable length columns to be pushed off-row so the effective row limit is higher than in previous releases of SQL Server [[http://technet.microsoft.com/en-us/library/ms186981(v=sql.105).aspx]]. I was able to create records of 24K.

The Large-Value Data Types [[http://technet.microsoft.com/en-us/library/ms178158(v=sql.105).aspx]] varchar(max), nvarchar(max), (text, ntext - deprecated naming) can take up to 2^{31} (30 for *n*) characters. They cannot be part of INCLUDE clause of a INDEX.

#13 - 04/03/2014 03:26 PM - Ovidiu Maxiniuc

- File temp-table-index.p added

- File novel.df added

- File find-index-limitation.p added

Write test cases which violate the Progress index size limit to determine error conditions we need to handle.

We define a table with some fields and a couple of indexes:

```
define temp-table tmp-novel
    field id as integer
    field chapter1 as character
    field chapter2 as character
    field chapter3 as character
    field chapter4 as character
    field chapter5 as character
    field chapter6 as character
    field chapter7 as character
    field a-int    as integer
    field a-int64  as int64
    field a-logical as logical
    field a-datetime as datetime
    index idx_content16 as unique
        a-datetime a-int a-int64 a-logical
        chapter1 chapter2 chapter3 chapter4 chapter5 chapter6
    index idx_chapter7 chapter7.
```

Giving increasing long string values to character fields:

```
tmp = chapter7.
repeat while true /* on error undo, leave */ :
    tmp = tmp + "x".

    assign chapter7 = tmp no-error.   /* in the case of the uni-fielded index, no validation is necessary */

    IF ERROR-STATUS:ERROR OR ERROR-STATUS:NUM-MESSAGES > 0 THEN DO:
        message "got error for " length(tmp)  length(chapter7).
        pause.
        leave.
    ENd.
end.
```

we obtain sooner or later the following error message:

```
The total length of the fields in an index exceeds max key size. Index <index-name> of table <database>.<table
> (129) (11353)
```

This looks like a fatal error, I was not able to catch it with no-error clause added to assign statement, and also not 'detectable' by on error phrase of the blocks.

Other observations:

- I tested the amount of bytes taken by different datatypes in a complex index and the remaining space for character fields depends on the other fields values (ie, int, int64, date, datetime, datetime-tz and logical take 2, 3 or up to 8 bytes depending on their actual value).
- From my tests, there is no difference if the table is permanent or temporary table.
- In the case of the indexes composed form multiple datatypes, validation is necessary to force p4gl to reindex. Strange enough, this is not necessary for the indexes with only character fields (one or many), in this case the index validation is done automatically.
- The max limit using 6 character fields components was 1965. With only one filed in index the the maximum size was 1970. Apparently one byte is lost for each additional index component.
- If the index is made INACTIVE, adding of records that brake the limitation is possible.
- If another index with same components is attempted to be created after the new records were added it will be silently discarded when applying changes to database.

Attached are test cases for permanent and temp-table testcases.

#14 - 04/03/2014 06:30 PM - Greg Shah

> This looks like a fatal error, I was not able to catch it with no-error clause added to assign statement, and also not 'detectable' by on error phrase of the blocks.

Make sure you also test ON ENDKEY, ON STOP and ON QUIT.

#15 - 04/04/2014 04:01 PM - Ovidiu Maxiniuc

My previous test cases were too complicated.
Thanks to Greg's suggestion, I used the other condition guards for the block. The ENDKEY and QUIT were ruled-out. Finally, our error proved to be a STOP condition. (The NO-ERROR option has no effect on STOP condition handling.)
Using the appropriate handling, the procedure can recover after the error and continue execution even if the message #11353 is forcefully printed (or displayed in a dialog in the case of GUI client).

Using the KISS principle I have simplified the usecase to:

```
DEFINE TEMP-TABLE tt1
   FIELD cf1 AS CHARACTER
   INDEX idx_cf1 cf1.

DEFINE VARIABLE cc AS CHARACTER INIT "a" NO-UNDO.

CREATE tt1.
REPEAT WHILE TRUE ON STOP UNDO, LEAVE:
   cc = cc + "a".
   cf1 = cc.
END.

DISPLAY "Detected index limit at " LENGTH(cf1).
```

The index limit is detected at 1971.

The 3rd observation from my previous note holds: if the index has other datatypes than characters, the record need to be validated to force Progress check the index dimension, otherwise the REPEAT loop became infinite (or at least until the cc string gets too large).

#16 - 04/04/2014 04:08 PM - Eric Faulhaber

Based on your findings of the longer index limit on their development systems, we have asked the customer to confirm which limit is in use in production environments. If it is also the longer one, we will have to revisit other possible approaches to the index problem.

#17 - 04/04/2014 04:25 PM - Ovidiu Maxiniuc

I started adding hint support for table fields. I added parsing and processing in TableHints.java, p2o.xml, hibernate.xml.
Next step is to generate the table ddls from hbm files. Most of the changes to not take care if the affected field is indeed of character type so the changes can serve for other datatypes (don't know if this will ever be the case) using dedicated rules.

A question that I should put earlier: how should the new hint look like?

```
<hints>
   <schema>
      <table name="hints4char">

         <!-- this fileld will be forced to have size of 30 characters in generated ddl -->
         <field name="char-field-30">
            <sql-size>30</sql-size>
         </field>

         <!-- this fileld will be forced to have size of 20 characters in generated ddl -->
         <field name="char-field-20">
            <sql size="20" />
            <phrase match="char" replace="some" />
         </field>

         <!-- this fileld will be forced to have size of 10 characters in generated ddl -->
         <field name="char-field-10" sql-size="10">
         </field>
```

At this moment, I work with the first solution (text content of sql-size as sub-element of field) but looking into the syntax of other hints I see that the second is preferred.
There is no big difference, but I would like to set this from the start and focus on more important part of the implementation.

#18 - 04/04/2014 04:31 PM - Eric Faulhaber

For the hint syntax, we generally follow the rule of thumb that if there can be only one value, the information is stored as an XML attribute of the element it describes, but if there can be an arbitrary number, the information is stored as a child element. So, in this case I think the third form is most appropriate, since a field cannot have multiple sizes.

#19 - 04/07/2014 03:50 PM - Ovidiu Maxiniuc

- File om_upd20140407a.zip added

Added update for review.
The implementation allows specifying sql-size attribute for all datatypes. However, only the sqlserver2008 and sqlserver2008 dialects will use the hint for generating the character ddl (varchar fields) with requested size. The fields that do not have the hint or the hint value is invalid, the default 8000 is used. Other dialects and other datatypes will ignore (for the moment) this hint.

#20 - 04/08/2014 02:18 PM - Ovidiu Maxiniuc

- File om_upd20140408a.zip added

Reuploaded the update after merging with latest bzr revision (10506).

#21 - 08/01/2014 09:53 AM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> Reuploaded the update after merging with latest bzr revision (10506).

Sorry I lost track of this. Does this update need separate review, or have these changes been incorporated into later updates in #2332?

#22 - 08/01/2014 09:59 AM - Ovidiu Maxiniuc

Eric Faulhaber wrote:

> Sorry I lost track of this. Does this update need separate review, or have these changes been incorporated into later updates in #2332?

IIRC, the changes in this update only handle sql-size hints and they were not added to any other update.

#23 - 08/01/2014 12:46 PM - Eric Faulhaber

Code review 20140408a:

Changes look good. Unfortunately (my fault), the update is now way out of date with the latest bzr revision and must be re-merged. My apologies. The result should only require conversion regression testing, not runtime.

Please merge the code up to the latest level, confirm that the sql-size hints still work after the merge and that you haven't regressed the extent denormalization feature, and of course make sure there are no conversion changes in the regression testing environment. Check with me before you commit and distribute, in case there are other pending updates in the meantime. I also would like to re-review after the merge, since it was difficult for me to isolate the changes when comparing against the latest code.

Vadim: please point Ovidiu to your extent denormalization test cases (should be checked into the testcases project) so he can manually regression test that feature after his merge.

#24 - 08/11/2014 05:12 PM - Ovidiu Maxiniuc

- File om_upd20140811b.zip added

I have finished merging the old update with bzr rev 10595. Please see the attached update.
My simple testcases were correct. I will check the extent denormalization before going to test.

#25 - 08/12/2014 10:38 AM - Eric Faulhaber

Code review 0811b:

Looks good, but I have some questions:

- In the event there is no sql-size hint (potentially a common use case), you now assign 0 to the length of the property (which we previously left unspecified) in the Hibernate mapping document. This document is not dialect-specific. What is the implication of specifying length for dialects other than SQL Server (e.g., PostgreSQL, for which we use the unbounded text data type)?
- In TableHints, you implemented generics for almost everything, but you skipped a few instance variables (e.g., the associations and indexes maps). Was this an oversight or intentional?

#26 - 08/12/2014 03:03 PM - Ovidiu Maxiniuc

- File om_upd20140812a.zip added

Eric Faulhaber wrote:

> Code review 0811b:
>
> Looks good, but I have some questions:
>
> - In the event there is no sql-size hint (potentially a common use case), you now assign 0 to the length of the property (which we previously left unspecified) in the Hibernate mapping document. This document is not dialect-specific. What is the implication of specifying length for dialects other than SQL Server (e.g., PostgreSQL, for which we use the unbounded text data type)?

This is a workaround to avoid a hibernate feature. If no sql length is specified in hbm file, it will automatically use 255 for all datatypes when calling dialect's getTypeName when computing the ddl. We cannot use 255 as a flag for value-not-set-use-full-size-of-8000 because if the size hint is set to 255 the sql will use the 8000. In other words, we cannot distinct between a no hint (no sql length) and an intentional 255 size-hint. Another solution would be to use directly the 8000 as the default sql length of all char fields (written into hbm files).
More details on javadoc of P2JSQLServer2008Dialect.getTypeName().

> - In TableHints, you implemented generics for almost everything, but you skipped a few instance variables (e.g., the associations and indexes maps). Was this an oversight or intentional?

I remember that I encounter some issues when processing these fields. I re-analyzed the code and the best generification looks like this:

```
    private final Map<Object, Object> associations = new HashMap<>();
    private final Map<String, Map> indexes = new HashMap<>();
```

That is because I cannot infer all the time the correct types or, in some cases different types of objects are store in the map.
For example indexes, we use indexes.put(indexName, map), when looking at map we can put in there two kind of data:
map.put(ATTR_UNIQUE, index.getAttribute(ATTR_UNIQUE)); the value is a String
and
map.put(KEY_INDEX_FIELDS, fieldsMap); the value is a Map<String, Map<String, String>> as inferred from the last lines of the constructor. I have done some additional code adjustments in the attached update:

- improved generics for TableHints (well, not very pleased with)

- recode the sql-size hints to be skipped for other datatypes, but for character where 0 is kept as size for MAX text fields in SQL2012 dialect.

The denormalization looks fine, apparently no harm was done during the upgrade of the code. I am going to put the update to quick conversion test.

#27 - 08/12/2014 04:01 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> Eric Faulhaber wrote:
>
> > Code review 0811b:
> >
> > Looks good, but I have some questions:
> >
> > - In the event there is no sql-size hint (potentially a common use case), you now assign 0 to the length of the property (which we previously left unspecified) in the Hibernate mapping document. This document is not dialect-specific. What is the implication of specifying length for dialects other than SQL Server (e.g., PostgreSQL, for which we use the unbounded text data type)?
>
> This is a workaround to avoid a hibernate feature. If no sql length is specified in hbm file, it will automatically use 255 for all datatypes when calling dialect's getTypeName when computing the ddl. We cannot use 255 as a flag for value-not-set-use-full-size-of-8000 because if the size hint is set to 255 the sql will use the 8000. In other words, we cannot distinct between a no hint (no sql length) and an intentional 255 size-hint.

Got it.

> Another solution would be to use directly the 8000 as the default sql length of all char fields (written into hbm files).

No, that would mean all hbm files would encode SQL Server-specific limits. I like the current approach of handling this in the dialect's getTypeName method and c'tor better.

> More details on javadoc of P2JSQLServer2008Dialect.getTypeName().
>
> > - In TableHints, you implemented generics for almost everything, but you skipped a few instance variables (e.g., the associations and indexes maps). Was this an oversight or intentional?
>
> I remember that I encounter some issues when processing these fields. I re-analyzed the code and the best generification looks like this:
> [...]
> That is because I cannot infer all the time the correct types or, in some cases different types of objects are store in the map.
> For example indexes, we use indexes.put(indexName, map), when looking at map we can put in there two kind of data:
> map.put(ATTR_UNIQUE, index.getAttribute(ATTR_UNIQUE)); the value is a String
> and
> map.put(KEY_INDEX_FIELDS, fieldsMap); the value is a Map<String, Map<String, String>> as inferred from the last lines of the constructor.

Sorry, that's due to my abuse of the pre-generics collections. Thanks for cleaning it up to the degree possible, short of rewriting it entirely.

> I have done some additional code adjustments in the attached update:
>
> - improved generics for TableHints (well, not very pleased with)
> - recode the sql-size hints to be skipped for other datatypes, but for character where 0 is kept as size for MAX text fields in SQL2012 dialect.
>
> The denormalization looks fine, apparently no harm was done during the upgrade of the code. I am going to put the update to quick conversion test.

OK, if it passes, please check this in and distribute it.

#28 - 08/13/2014 04:35 PM - Ovidiu Maxiniuc

The conversion test was successful, there are no changes in the ddl code, only the .hbm.xml are affected: and extra length="0" for the character fields because of the workaround from first issue discussed in previous notes.

Although a runtime regression test is not mandatory, I started one, but it failed multiple times.
The cause could be (org.eclipse.jetty.util.component.AbstractLifeCycle:WARNING) FAILED ServerConnector@67617c3c{SSL-HTTP/1.1}{0.0.0.0:7563}: java.net.BindException: Address already in use

Also, I wonder if I should delay committing this until after fixing the extent denormalization issue / #2134 / #2376 ?

#29 - 08/14/2014 12:28 PM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> The cause could be (org.eclipse.jetty.util.component.AbstractLifeCycle:WARNING) FAILED ServerConnector@67617c3c{SSL-HTTP/1.1}{0.0.0.0:7563}: java.net.BindException: Address already in use

Yes, the previous server instance was still running, with Jetty still using port 7563, so the new server instance could not start.

#30 - 09/09/2014 01:29 PM - Ovidiu Maxiniuc

The update passed the regression test and was committed to bzr (rev 10599) and distributed by mail on 14 aug 2014.

#31 - 03/25/2016 01:33 PM - Eric Faulhaber

- % Done changed from 0 to 100

- Status changed from WIP to Closed

The features in this issue have been implemented, but not heavily tested. Bugs coming out of this will be treated as separate issues.

#32 - 11/16/2016 12:06 PM - Greg Shah

- Target version changed from Milestone 11 to Cleanup and Stablization for Server Features

#33 - 01/19/2017 06:10 PM - Greg Shah

- Related to Feature #2312: match Progress collation in a SQL Server database added

Files

novel.df	2.2 KB	04/03/2014	Ovidiu Maxiniuc
find-index-limitation.p	1.94 KB	04/03/2014	Ovidiu Maxiniuc
temp-table-index.p	2.72 KB	04/03/2014	Ovidiu Maxiniuc
om_upd20140407a.zip	48 KB	04/07/2014	Ovidiu Maxiniuc
om_upd20140408a.zip	48.4 KB	04/08/2014	Ovidiu Maxiniuc
om_upd20140811b.zip	53.9 KB	08/11/2014	Ovidiu Maxiniuc
om_upd20140812a.zip	53.9 KB	08/12/2014	Ovidiu Maxiniuc