Database - Feature #3010

retrieve new primary keys from database in batch

03/01/2016 10:39 AM - Eric Faulhaber

Status:	WIP	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	60%
Category:		**Estimated time:**	0.00 hour
Target version:	Performance and Scalability Improvements		
billable:	No	**vendor_id:**	GCD

Description

History

#1 - 03/01/2016 10:54 AM - Eric Faulhaber

I've noticed that in some use cases, we are spending a lot of time retrieving new primary keys from the database, using the p2j_id_generator_sequence. This happens when we run out of recovered keys in the IdentityPool.

We could do a better job retrieving batches of these in anticipation of future requests. Probably the simplest implementation is for SequenceIdentityManager to spawn a background thread to pre-fetch some number of these, and add them to the pool, so that they are quickly available when needed. The amount to pre-fetch would be configurable in the directory, but would default to some sensible number. We would do this periodically, when the pool reaches some threshold of remaining keys.

Something to look into in this context is whether we can speed up the call to nextval on the sequence, if multiple of these are made together. Can we use JDBC batch mode for this, or a stored procedure?

#2 - 03/01/2016 11:10 AM - Eric Faulhaber

Implementation note: since reclaimed keys are stored individually in the IdentityPool, we would need a new API in that class to support adding keys in contiguous ranges. This will allow us to minimize memory consumption from batches of fresh keys added by this new mechanism.

#3 - 03/01/2016 11:23 AM - Eric Faulhaber

It also seems our implementation to get and reclaim primary keys for temp-tables is overly complicated. See TemporaryBuffer methods nextPrimaryKey and reclaimKeys. I don't think it's necessary to divvy up the primary keys by DMO interface. Having a single id counter for all the temp-tables in a particular context should be sufficient. This will require fewer lookups and less memory for a flatter data structure. We still shouldn't share the id counters and reclaimed keys across contexts, as that will require additional synchronization, for no good benefit.

#4 - 04/21/2016 01:36 PM - Eric Faulhaber

Having done more profiling since I entered this issue, I think the primary use comes from the metadata LockTableUpdater, which makes its updates on a single thread. I think this class may be using the ID generator sequence of the primary database when it doesn't need to. I need to confirm, but if so, we can simply bypass the use of the sequence here with a simple counter and avoid these calls to the database altogether.

#5 - 09/07/2016 11:40 PM - Eric Faulhaber

I've implemented this, but the solution requires a dialect-specific query. For PostgreSQL, I use:

```
select nextval('p2j_id_generator_sequence') from generate_series(?, ?)
```

I expect something similar will work for H2:

```
select nextval('p2j_id_generator_sequence') from system_range(?, ?)
```

I don't yet have a solution for SQL Server.

Ovidiu, can you suggest one? It has to be fast, as the whole point of this is to improve performance compared to making multiple round trips to the database for sequence values. The values returned do not have to be contiguous; that is, it is OK if another connection hits the sequence while we are executing this query.

In the absence of this, SequenceIdentityManager will default to the existing, single key retrieval approach.

#6 - 09/07/2016 11:41 PM - Eric Faulhaber

- Status changed from New to WIP

- % Done changed from 0 to 60

#7 - 09/08/2016 09:39 AM - Ovidiu Maxiniuc

Eric Faulhaber wrote:

> Ovidiu, can you suggest one? It has to be fast, as the whole point of this is to improve performance compared to making multiple round trips to the database for sequence values. The values returned do not have to be contiguous; that is, it is OK if another connection hits the sequence while we are executing this query.

Eric,
At first I did not get what's the new syntax used for. I got it now, we need to reserve a set of N values ahead from the sequence for later use.

I dig into MS SQL docs and found that this is possible for this dialect. However, there is a catch: they do it in their own way. The SQL Server dialect does not receive the result as a normal query, but as a subsequence described by output parameters of sys.sp_sequence_get_range function. Practically: one need to invoke the function with sequence name and subsequence length (ie the number of returned elements in sequence) and provide the output parameters for the metadata describing the result. The first returned value is mandatory. If the sequence does not cycle you could compute the last reserved value (the values are guaranteed to be contiguous). If it does, things get messy but it is supposed you will decode the result using the other optional output parameters: cycle-count, increment, min and max values.

As conclusion I don't think this can be used like the other dialects. Probably using a sql native UDF that will convert the sub-sequence metadata into a table. I will try to do that as soon as possible. Please let me know the priority of this issue (from MSSQL POV) so I can plan temporarily work on Windows machine. I estimate about 4 hours for this task for a complete solution.

#8 - 09/08/2016 11:01 AM - Eric Faulhaber

I was hoping there would be a simple analogue to the query approach I can use with the other dialects. Is something like this (http://sqlperformance.com/2013/01/t-sql-queries/generate-a-set-1) of any use?

This is not high priority. The current mechanism works fine as a fallback.

#9 - 11/16/2016 12:30 PM - Greg Shah

- Target version changed from Milestone 17 to Performance and Scalability Improvements