Database - Bug #4917

eliminate redundant ORDER BY elements in multi-table queries

09/27/2020 02:55 PM - Eric Faulhaber

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

Description

Related issues:

Related to Database - Feature #4030: improve CompoundQuery optimizer	**New**
Related to Database - Bug #4931: possible ProgressiveResults performance impr...	**New**

History

#1 - 09/27/2020 03:08 PM - Eric Faulhaber

We have found that in certain cases (e.g., optimized CompoundQuery), multi-table queries are generated with ORDER BY clauses containing redundant elements. This can severely impact query performance, by causing the database's query planner to select an inefficient plan, which does not take advantage of available indices.

Example:

```
for each person no-lock,
    each pers-addr
         where pers-addr.site-id = person.site-id
         no-lock:

   display person.last-name
           person.emp-num
           person.site-id
           pers-addr.addr-id.

end.
```

This is converted to (relevant portions only):

```
CompoundQuery query0 = new CompoundQuery();
...
query0.initialize(false, false);
query0.addComponent(new AdaptiveQuery().initialize(person, ((String) null), null, "person.siteId asc, person.e
mpNum asc", LockType.NONE));
query0.addComponent(new AdaptiveQuery().initialize(persAddr, "persAddr.siteId = ?", null, "persAddr.siteId asc
, persAddr.empNum asc", new Object[]
{
   new FieldReference(person, "siteId")
}, LockType.NONE));
```

At runtime, this query is optimized to a server side join in a single AdaptiveQuery. The FQL is:

```
select person, persAddr
from Person__Impl__ as person,
PersAddr__Impl__ as persAddr
where (persAddr.siteId = person.siteId)
order by person.siteId asc, person.empNum asc, persAddr.siteId asc, persAddr.empNum asc, persAddr.id asc
```

The ORDER BY element persAddr.siteId asc is redundant, because the WHERE clause in the joined query ensures these values are the same value in both tables for every row.

This is not the best example, because even if the redundant element is identified and removed, the remaining elements persAddr.empNum asc, persAddr.id asc may still cause a problem with the query plan. Furthermore, these sample tables are typically so small that a table scan query plan will be selected anyway. However, there are cases using larger tables where the removal of the redundant element makes a huge difference in the performance of the query.

#2 - 09/27/2020 03:15 PM - Eric Faulhaber

I initially thought there would be an AbstractJoin object in the second query component we could use to analyze the join and remove the redundant ORDER BY element. However, my recollection of how this optimization worked was incorrect.

As we can see in the converted code above, there is no AbstractJoin instance in this case. The conversion from client-side to server-side join is achieved by converting FieldReference query substitution parameters to their representative strings, and replacing the substitution placeholders with those strings. For instance, given an outer query component on the Person DMO and an inner component on the PersAddr DMO, the client-side joining where clause of:

```
persAddr.empNum = ?
```

with a query substitution parameter of new FieldReference(person, "empNum"), the parameter is dropped and the where clause snippet becomes:

```
persAdd.empNum = person.empNum
```

Doing it this way is fast, as there is no analysis of the match expression required, but now that we need some more context about that match, we don't have enough information to determine whether an ORDER BY element is redundant, without some analysis of the where clause. This probably will need to be done and exposed by HQLPreprocessor.

This analysis was done only for the optimized CompoundQuery case, but I expect there are cases where we create multi-table queries during static or dynamic query conversion with this same defect.

#3 - 09/27/2020 03:44 PM - Greg Shah

- Related to Feature #4030: improve CompoundQuery optimizer added

#4 - 09/27/2020 03:52 PM - Constantin Asofiei

Something to share about my investigations with H2 - if the chosen index didn't match the ORDER BY clause, then H2 will sort the records again; and this was expensive especially for LIMIT 1 queries.

My point here is to ensure that by removing a field from the ORDER BY clause, we are still on the right index; and also this will not trigger a secondary sort of the result.

#5 - 09/27/2020 04:09 PM - Eric Faulhaber

This fix is intended only for the case where we have sort clauses for more than one table concatenated together, and would always leave the part taken from the first component intact. The sort phrase for each query component represents the selected index for that component's record phrase. The first part of the combined phrase (unless prepended with elements from a converted BY clause) will always be the most important in terms of the database's query planner selecting an index. It would not be altered, since there is no "outer" table with which it is joined, and therefore, no element in that first part can be redundant with something that came before. There can be no index in any table of the join which contains elements from all the tables, so altering the ones further in **I think** can only improve or have no effect on the query plan.

#6 - 09/29/2020 12:51 PM - Eric Faulhaber

- Related to Bug #4931: possible ProgressiveResults performance improvement added

#7 - 10/01/2020 10:38 AM - Ovidiu Maxiniuc

- File Dropped_redundant_ORDER-BY_elements.patch added

Eric,
I have an implementation which is quite stable for my test-case. I am putting it to the test against hotel_gui and customer's application.
The idea was to rewrite PreselectQuery.assembleHQL() so that the ORDER BY clause to be generated after preprocessing the fql to let the HQLPreprocessor to identify the redundant properties and removed them from the sort criteria list. Finally, the clause is generated the last one, based on the list that was simplified.

I attached the patch, if you have time, please review. If everything is fine, I will push it to 3821c.

#8 - 10/01/2020 11:33 AM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> I attached the patch, if you have time, please review. If everything is fine, I will push it to 3821c.

Maybe I didn't read the patch correctly... does this still work if the fields being joined have different names in the inner and outer table?

#9 - 10/01/2020 11:44 AM - Ovidiu Maxiniuc

I did not make any test regarding this, but it should work.
If we have A.a1 = B.b1 as join relation, the logic that drops the redundant criterion does not check whether a1 = b1. Instead it takes the list of criteria (A.a1, A.a2, B.b1, B.b2, ..) and the subst replacement pairs ((A.a1, B.b1), (...)) computed by HQLPreprocessor.inlineReferenceSubs() and if finds A.a1 and B.b1 in the criteria list, it will drop the second criterion (in this case B.b1). The resulting list (A.a1, A.a2, ...) is used to create the ORDER BY option.

#10 - 10/01/2020 11:45 AM - Ovidiu Maxiniuc

I did not log any of this activity. I am thinking of doing it.

#11 - 10/01/2020 11:54 AM - Eric Faulhaber

Ovidiu Maxiniuc wrote:

> I did not log any of this activity. I am thinking of doing it.

If you mean permanent logging (as opposed to temporary logging just for debugging as you develop this feature), please only check isLoggable once at class load time and cache that value in a static boolean flag, so as to avoid the overhead of the more dynamic check at every log point.

#12 - 10/01/2020 12:10 PM - Ovidiu Maxiniuc

I do not see any issues with customer code.
The query that was exceeding 150ms is no more present in my statistics but a variant of it, having the offset along the limit option. It is timed very similar (175.89ms - 186.245ms). I investigated and their values are: limit 13825 offset 600. I executed the query using psql and I get much bigger times. OTOH, using these options, the query returns 1900 rows. I am not sure if this time can be improved.

#13 - 10/01/2020 01:16 PM - Ovidiu Maxiniuc

I did not observe any regressions while testing hotel_gui and customer application so I decided to commit the update.
The revision is 3821c/11652.

Files

Dropped_redundant_ORDER-BY_elements.patch	33.5 KB	10/01/2020	Ovidiu Maxiniuc