Database - Bug #7321

Improve FWD-H2 wildcard projection

05/02/2023 09:28 AM - Alexandru Lungu

Status:	Test	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Dănuț Filimon	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD	**version:**	

Description

History

#1 - 05/02/2023 09:56 AM - Alexandru Lungu

Recently, queries over the temporary database use the wildcard syntax: tt.* to select all the columns. If there are computed columns, they are marked with INVISIBLE at table creation.

This task is meant to optimize the tt.* syntax. The initial intend was to lower the parsing time, as this is way shorter than a ~1k character field list. However, this wildcard is extended into a projection of fields. H2 generates a list of ExpressionColumn for each tt field, so that tt.* wildcard is replaced with a list of expressions. Therefore, the query engine will project a tt row into the list of ExpressionColumn. This is a performance and memory issue. Note that such statements are cached, so storing ~100 ExpressionColumn per entry in the psCache is not ideal.

There are 2 hostpots here:

- Select.expandColumnList which replaces TT.* into a [PUBLIC.TT.F1, PUBLIC.TT.F2] array.
- The LazyResultQueryFlat.fetchNextRow which calls ExpressionColumn.getValue for each column individually. In all cases, the table row will match exactly the list of expression columns

Currently in FWD-H2:

```
Value[] row = new Value[columnCount];
for (int i = 0; i < columnCount; i++) {
   Expression expr = expressions.get(i);
   row[i] = expr.getValue(getSession());
}
```

I don't think this can be handled uniformly for any Select; invisible columns and joins may break the expected order of the projection. However, one can check that expressions.size() == 1 && expressions.get(1) instanceof Wildcard. In this case, do a System.arraycopy from the search row into the final row value array.

For profiling, please use something like:

```
stmt.executeUpdate("create table tt(f1 int, f2 int)"); // maybe ~100 columns
stmt.executeUpdate("insert into tt values (1, 2)"); // maybe ~1000 rows
stmt.executeQuery("select tt.* from tt"); // maybe 100 runs
```

#2 - 05/02/2023 10:03 AM - Alexandru Lungu

- Status changed from New to WIP

- Assignee set to Dănuț Filimon

#3 - 05/05/2023 05:43 AM - Alexandru Lungu

On a large customer application, I checked how much time this scenario (wildcard expansion and search row projection) takes:

Scenario	Expand time (ms)	Expand count	Projection time (ms)	Projection count
Server start-up	5.988	1,329	47.639	28,969
Start-up and run-time	17.693	4,727	269.058	129,778

It looks like the expand time is not that problematic, but the projection time is really demanding. This is because the expansion happens only at prepare time. Projection is happening at each row fetched.
Therefore, we can let the expansion happen. This will save us from possible regressions whenever expressions is used and an wildcard is not expected. Consider improving only the projection time. Use something like originalExpressions.

#4 - 05/09/2023 02:56 AM - Dănuț Filimon

Committed 7321a_h2/rev.18. Improved wildcard projection by doing a fast copy of the current search row.

This improvement works only for select statements that do not use EXCEPT (columns that should not be retrieved are specified, a copy of the row can't be done), GROUP BY or **JOINS**. I did a profiling test which consisted of a table with 100 character fields of length 30, 5000 initial creates and 2500 select statements using wildcard, the results were:

	7321a_h2.rev17 (ms)	7321a_h2.rev18 (ms)	Difference (%)
Test (Avg of 5)	16620.2	11155.6	-32.873%

I also tested a customer application for regressions (~20 min) and I had no issues. Please review.

#5 - 05/11/2023 10:38 AM - Alexandru Lungu

- % Done changed from 0 to 80

Review of 7321a_h2/rev.18

- I am not sure if you actually need currentSearchRow or current. Anyway, your TableFilter.getCurrentRow null checks currentSearchRow and returns cursor.get(). For consistency, this should have been cursor.getSearchRow(). As I said however, I am not sure if you actually need the search row or the current row. According to Cursor javadoc, the search row has only the indexed fields used for look-up. My best guess here is that H2 uses the search row as much as it can to resolve columns to avoid computing current. If current is computed, it is used directly. Therefore, it looks to me that TableFilter.get is exactly what you need, instead of your custom TableFilter.getCurrentRow. Please do the change and do some investigation on this.
- I am OK with the other changes.

I will do a quick test and profile and merge. This looks safe and fast enough

#6 - 05/12/2023 03:38 AM - Dănuț Filimon

Committed 7321a_h2/rev.19. Removed TableFilter.getCurrentRow(), used TableFilter.get() instead.

Instead of using the SearchRow I used a Row because I could make use of Row.getValueList(). The values from the search row can be retrieved one by one using getValue(int) so I opted for Row since it already had the necessary method for getting all the values from the row.

Alexandru Lungu wrote:

> According to Cursor javadoc, the search row has only the indexed fields used for look-up ... Therefore, it looks to me that TableFilter.get is exactly what you need ...

When the select statement uses the IndexCursor, only the indexed fields will be available. I remarked this only when using a persistent H2 database. As you said, TableFilter.get() is needed in this case.

#7 - 05/24/2023 05:40 AM - Alexandru Lungu

- Status changed from WIP to Review

- % Done changed from 80 to 100

I am OK with the changes.
I also included some improvements on calling Select.isEverything faster (iterate the expressions only in very specific cases if wildcard is used).
Merged 7321a_h2 to FWD-H2 trunk as rev. 19
Danut, please update your FWD-H2 trunk and review rev. 19. Thank you!

Archived 7321a_h2.

#8 - 05/29/2023 02:41 AM - Alexandru Lungu

- Status changed from Review to Test