Database - Bug #7047

Problem with AdaptiveQuery and ScrollableResults

01/18/2023 04:03 AM - Igor Skornyakov

Status:	New	**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 Database - Feature #7061: Enable the use of lazy result sets in H2	**Closed**

History

#1 - 01/18/2023 04:04 AM - Igor Skornyakov

It looks like we have problems either with AdaptiveQuery or ScrollingResults (or both).
Consider the attached program fill.p. It uses permanent table ttsrc (see attached definition and contents). Please note that the table has no index.

The test runs OK with H2 and MariaDB but with PostgreSQL, it fails will the exception:

```
org.postgresql.util.PSQLException: Operation requires a scrollable ResultSet, but this ResultSet is FORWARD_ON
LY.
        at org.postgresql.jdbc.PgResultSet.checkScrollable(PgResultSet.java:280)
        at org.postgresql.jdbc.PgResultSet.first(PgResultSet.java:355)
        at com.mchange.v2.c3p0.impl.NewProxyResultSet.first(NewProxyResultSet.java:815)
        at com.goldencode.p2j.persist.orm.ScrollableResults.execute(ScrollableResults.java:434)
        at com.goldencode.p2j.persist.orm.ScrollableResults.first(ScrollableResults.java:139)
        at com.goldencode.p2j.persist.ScrollingResults.first(ScrollingResults.java:134)
        at com.goldencode.p2j.persist.ResultsAdapter.first(ResultsAdapter.java:131)
        at com.goldencode.p2j.persist.PreselectQuery.first(PreselectQuery.java:2457)
        at com.goldencode.p2j.persist.AdaptiveQuery.first(AdaptiveQuery.java:1228)
        at com.goldencode.p2j.persist.PreselectQuery.first(PreselectQuery.java:2424)
        at com.goldencode.p2j.persist.QueryWrapper.lambda$first$0(QueryWrapper.java:1892)
        at com.goldencode.p2j.persist.QueryWrapper.handleQueryOffEnd(QueryWrapper.java:6884)
        at com.goldencode.p2j.persist.QueryWrapper.first(QueryWrapper.java:1892)
```

I understand that the reason is the following:
The AdaptiveQuery.executeQuery() instead of creating ProgressiveResults which seems to be most appropriate here, calls PreselectQuery.executeQuery creating ScrollingResults because probablyRequiresResort() returns true.
In addition ScrollingResults.first() calls ResultSet.first() even if it is the first call after executeQuery. This is OK for H2 and MariaDB, but not for PostgreSQL.

If we add an explicit ordering to the query or an index to the ttsrc table the test runs with PostgreSQL as well

#2 - 01/18/2023 04:06 AM - Igor Skornyakov

- File ttsrc.d added

- File ttsrc.df added

- File fill.p added

Added attachements

#3 - 02/07/2023 07:23 AM - Alexandru Lungu

- Related to Feature #7061: Enable the use of lazy result sets in H2 added

#4 - 02/21/2023 09:34 AM - Alexandru Lungu

- Start date deleted (01/18/2023)

As a side investigation which may help here:

AdaptiveQuery is not considering recid index as a sorting index. Therefore, probablyRequiresResort returns true for non-indexed queries, just like in #7047-1. This is something that should be fixed separately.
We always add a physical tree index for (multiplex, recid) pair, but our code just searches through the legacy indexes and thus a ScrollingResults is used. ProgressiveResults requires more result-sets, so the example in #7047-1 may be in luck - another query is used to retrieve the first record, so there is no actual ResultSet.first done which could have caused problems.

I can't say why PostgreSQL behaves differently, but logically all non-scrolling 4GL queries should be TYPE_SCROLL_INSENSITIVE. In 4GL non-scrolling doesn't mean one single way of iteration (FORWARD_ONLY), it means that we can't reposition, but we still have the ability to get back to some previous results: using GET PREVIOUS or GET FIRST. My point here is that we should use TYPE_SCROLL_INSENSITIVE for all AdaptiveQuery instances (and maybe PreselectQuery?). Anyways, a performance investigation for TYPE_SCROLL_INSENSITIVE vs FORWARD_ONLY will be done in #7061 anyways.

#5 - 02/21/2023 11:57 AM - Eric Faulhaber

Alexandru Lungu wrote:

> I can't say why PostgreSQL behaves differently, but logically all non-scrolling 4GL queries should be TYPE_SCROLL_INSENSITIVE. In 4GL non-scrolling doesn't mean one single way of iteration (FORWARD_ONLY), it means that we can't reposition, but we still have the ability to get back to some previous results: using GET PREVIOUS or GET FIRST. My point here is that we should use TYPE_SCROLL_INSENSITIVE for all AdaptiveQuery instances (and maybe PreselectQuery?). Anyways, a performance investigation for TYPE_SCROLL_INSENSITIVE vs FORWARD_ONLY will be done in #7061 anyways.

Be **very careful** about changing the scroll type. I intentionally biased it toward TYPE_FORWARD_ONLY as much as possible, to address OOME problems we were having early on in FWD development, with PostgreSQL on the back end. This may not be logically the best match for 4GL behavior, but not crashing the server JVM is a pretty important consideration ;)

When using PostgreSQL, TYPE_FORWARD_ONLY (along with a non-zero JDBC fetch size) lets us use a server-side database cursor to fetch the results. This is important when fetching large result sets, so we don't pull the entire result set into the JDBC driver's memory and cause OOME. We addressed this problem for PostgreSQL by setting fetch size > 0 (but not to a huge value). Note the comments in the Persistence.list API javadoc about preferring scroll to list in this regard. There probably should be similar warnings in the scroll methods' javadoc about the scroll type.

See also https://jdbc.postgresql.org/documentation/query/.

I don't know what other dialects do, in terms of the default JDBC driver behavior on this point, nor am I aware of the conditions needed for server-side cursors to be used in other databases. This is something we should understand, as the API currently is based on PostgreSQL's behavior in this area.

Files

fill.p	498 Bytes	01/18/2023	Igor Skornyakov
ttsrc.df	457 Bytes	01/18/2023	Igor Skornyakov
ttsrc.d	268 Bytes	01/18/2023	Igor Skornyakov