Article
Benjamin De Boe · Mar 25, 2020 5m read

New in 2020.1: the Universal Query Cache

InterSystems IRIS 2020.1 brings a broad set of improved and new capabilities to help build important applications. In addition to the many significant performance improvements accrued through 2019.1 and 2020.1, we are introducing one of our biggest changes in recent SQL history: the Universal Query Cache. This article provides more context on its impact to SQL-based applications at a technical level.

What is this UQC?

Besides a possible game-changer when used as an acronym in Scrabble, the Universal Query Cache is meant to simplify SQL operations across the board by managing all types of SQL statements in a single cache. Previously, statements issued as dynamic SQL (using `%SQL.Statement` infrastructure) or through JDBC and ODBC were already compiled into optimized code as a cached query class at "prepare" time. When using embedded SQL, on the other hand (using `&SQL()` syntax), the code to execute the query was generated inline and remained part of the application. This resulted in embedded SQL queries being treated quite differently to all other forms of SQL query that IRIS supports.

A cached query class lives independently of the code that issued the statement and can be purged and re-generated as needed, for example to end up with a more efficient access path based on updated table statistics, after a new index was added or after upgrades (see also this note on frozen query plans on upgrade). Embedded SQL, on the other hand, used to be static and could not automatically take advantage of new table statistics and indices.

In 2020.1, with the Universal Query Cache, we're also generating cached query classes for embedded SQL and no longer mix this code into the same classes holding application logic. The most important benefit is that embedded SQL will now no longer require source classes to be recompiled to take advantage of more recent table statistics or new indices. Any time a table referenced in the UQC is updated we will automatically generate a new cached query class to make sure it is running the most efficient code we can generate. As we generate the embedded SQL into a class, we can take advantage of faster cursor variable storage that all other queries were already using. This means that queries that do a lot of work will run faster than before. In addition, UQC removes all SQL dependencies between classes which could often cause developer headaches trying to manage which class needed to be compiled before which other class.

This is a big change and while we worked hard and did extensive testing to anticipate many "creative" setups (deployed code, namespace switches inside application code, ⋯), there may be yet more creativity out there than we could think of. Therefore, please pay extra attention when upgrading applications that make heavy (and possibly creative!) use of embedded SQL to 2020.1 and please get in touch through the WRC if you notice anything out of the ordinary.

Are embedded and dynamic SQL now one and the same?

There's still going to be a somewhat semantic difference between dynamic and embedded SQL in terms of usage. Embedded SQL works with named cursors, which are a little clumsier than working with an actual object handle to the query in Dynamic SQL. But that's largely a matter of personal taste and there is obviously no reason to rewrite an application from the one flavor into the other, just for the fun of it.

As for performance, embedded SQL used to have a reputation for being faster than dynamic SQL, but much of that difference has evaporated over the past few years thanks to improvements in code generation and object handling. Still, values from an embedded SQL cursor can be fetched straight into a variable, which is a tiny bit faster than accessing them through an object property in the dynamic SQL equivalent. It is therefore still faster, but only by a

very small margin, and it may not outweigh (subjective!) development convenience in many cases.

The fine print

The introduction of UQC means embedded SQL in 2020.1 has a very small fixed overhead it did not have before, namely calling a different class method in the cached query object behind the scenes to run the query, rather than staying in the same routine or class context. The work in generating and compiling the cached query code, previously part of compiling the class or routine containing `&SQL()` constructs, now happens the first time the query is invoked, as it does with dynamic SQL and typically taking advantage of more realistic table statistics. Note that developers can force compiling embedded SQL using the new `/compileembedded=1` compile flag.

Separately, there is a variable benefit from the use of fast i%var access in the generated cached query code to hold cursor state information, which for most nontrivial queries will offset the fixed overhead and in fact make embedded SQL faster compared to previous versions. Only very small statements (e.g. dead-simple INSERTs) may experience a performance regression compared to 2019.4. Our own experiments put this at about 6% for such simple queries, vs a 3% performance *increase* for complex queries.

Customers upgrading from versions earlier than 2019.4 are unlikely going to see anything other than performance *improvements* though. Between 2019.1 and 2019.4, we introduced a lot of other performance enhancements across the SQL and kernel layers that will outweigh the tiny fixed overhead we had to introduce for UQC by a fair margin. Please see this GS session recording for an overview of many of those changes.

Conclusion

In short, the Universal Query Cache will make a DBA ' s life easier by ensuring they can quickly refresh query plans to take into account current statistics and available indices. Especially when deploying SQL application to different customer environments, this is an important enhancement that gives complete freedom to developers to pick the style of SQL that fits them best.

Article based on valued contributions from @Mark Hanson *and* @Tom Woodfin

#SQL #InterSystems IRIS