Article
Benjamin De Boe · Dec 15, 2021 4m read

2021.2 SQL Feature Spotlight - Smart Sampling & Automation for Table Statistics

This is the second piece in our series on 2021.2 SQL enhancements delivering an adaptive, high-performance SQL experience. In this article, we'll zoom in on the **innovations in gathering Table Statistics**, which are of course the primary input for the Run Time Plan Choice capability we described in the previous article.

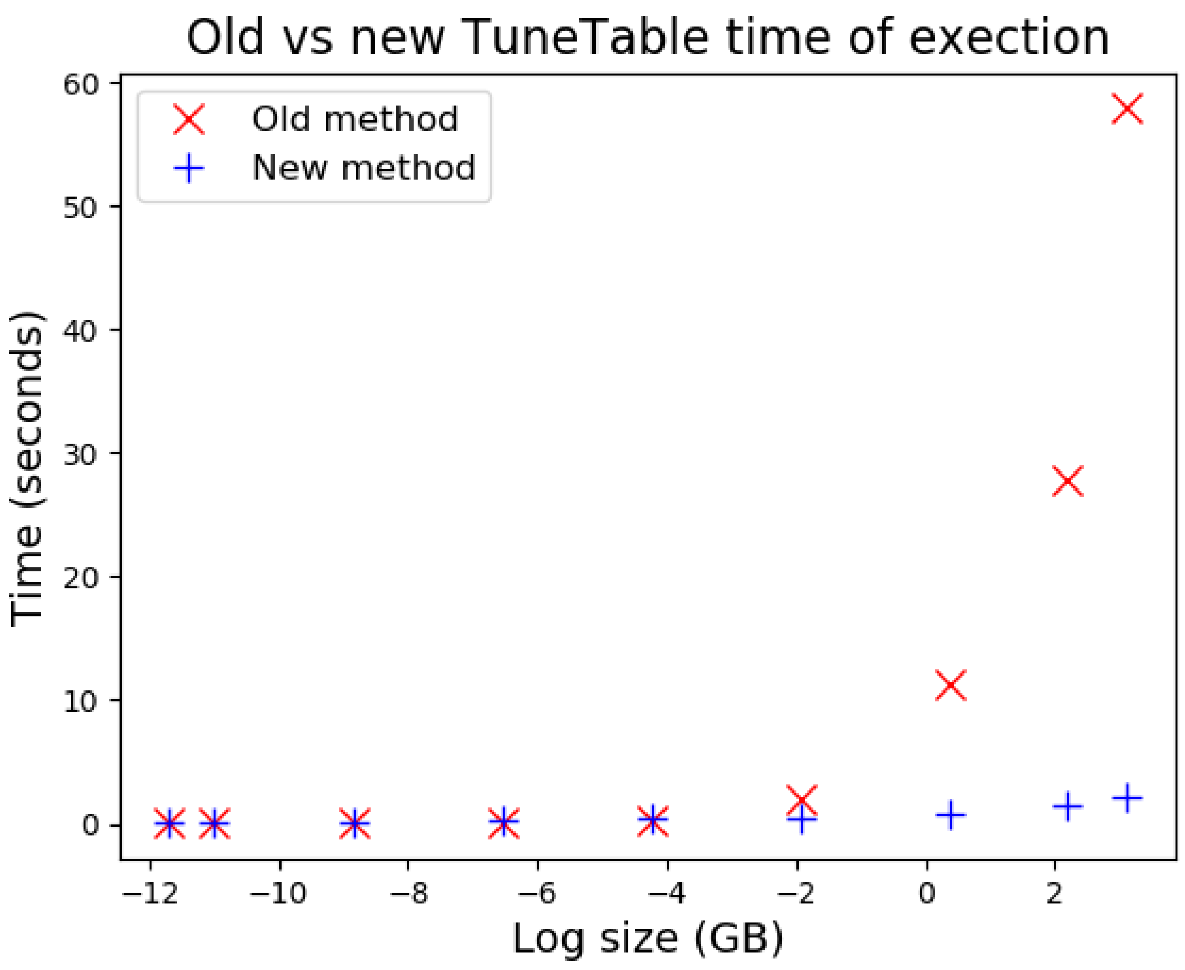
You've probably heard us say this many times: Tune your Tables!

Tuning your tables through the TUNE TABLE SQL command or $SYSTEM.SQL.Stats.Table ObjectScript API means collecting statistics on your table's data to help IRIS SQL figure out a good query plan. Those statistics include important information such as the approximate number of rows in the table, which helps the optimizer decide on things such as JOIN order (starting with the smallest table is usually most efficient). Many calls to InterSystems support about query performance can be addressed by simply running TUNE TABLE and giving it another try, as running the command will invalidate existing query plans so the next invocation picks up the new statistics. From those support calls, we see two recurring reasons why those users hadn't collected table statistics already: they didn't know about them, or they couldn't afford the overhead of running them on the production system. In 2021.2, we've addressed both.

Block-level Sampling

Let's start with the second one: the cost of collecting statistics. It's true that collecting table statistics may cause a fair amount of I/O and therefore overhead *if* you are scanning the entire table. The API already supported sampling only a subset of rows, but still the operation had quite a reputation for cost. In 2021.2, we've changed this to no longer select random rows through looping through the master map global, but immediately reach out to the physical storage underneath and let the kernel take a random sample of the raw database blocks for that global. From those sampled blocks, we infer the SQL table rows they store and proceed with our regular per-field statistics-building logic.

You can compare this to going to a large beer festival and rather than walk through all the aisles and choose a few brewery stands from which you put a bottle each in your cart, you just ask the organizers to give you a crate with random bottles and save yourself the walk. (in this beer tasting analogy, the walk would actually be good idea). To sober up, here is a simple graph plotting the old row-based approach (red crosses) against the block-based approach (blue crosses). The benefits are huge for large tables, which happens to be the ones some of our customers were wary about running TUNE TABLE on...



There are a small number of limitations to block sampling, the most important being that it can't be used on tables that dwell outside of the default storage mappings (e.g. projecting from a custom global structure using %Storage.SQL). In such cases, we'll still revert to row-based sampling in exactly the way it worked in the past.

Automatic Tuning

Now that we've got that overhead perception issue out of the way, let's consider the other reason some of our customers weren't running TUNE TABLE: they didn't know about it. We could try to document our way out of it (and acknowledge there's always room for better docs), but decided this super-efficient block sampling actually provides an opportunity to do something we've long wanted: just automate the whole thing. Starting with 2021.2, when you're preparing a query against a table for which there are no statistics available at all, we'll first use the above block sampling mechanism to collect those statistics and use those for query planning, saving the statistics back to the table metadata so they can be used by onward queries.

While that may sound scary, the chart above shows how this statistics gathering work only starts taking on the order of seconds for GB-sized tables. If you're using an inappropriate query plan to query such a table (because of a lack of proper statistics), that's likely going to be far more costly than doing the quick sampling upfront. Of course, we'll only do this for tables where we can use block sampling and (sadly) proceed without statistics for those special tables that only support row-based sampling.

As with any new feature, we're eager to hear about your initial experiences and feedback. We have further ideas around automation in this area, such as keeping statistics fresh based on table usage, but would love to make sure those are grounded in experiences from outside the lab.

#Relational Tables #SQL #InterSystems IRIS