Article
Jean Millette · Aug 22, 2019 3m read

A Case for Thawing Frozen Query Plans After Upgrade

Our team is reworking an application to use REST services that use the same database as our current ZEN application. One of the new REST endpoints uses a query that ran very slowly when first implemented. After some analysis, we found that an index on one of the fields in the table greatly improved performance (a query that took 35 seconds was now taking a fraction of a second).

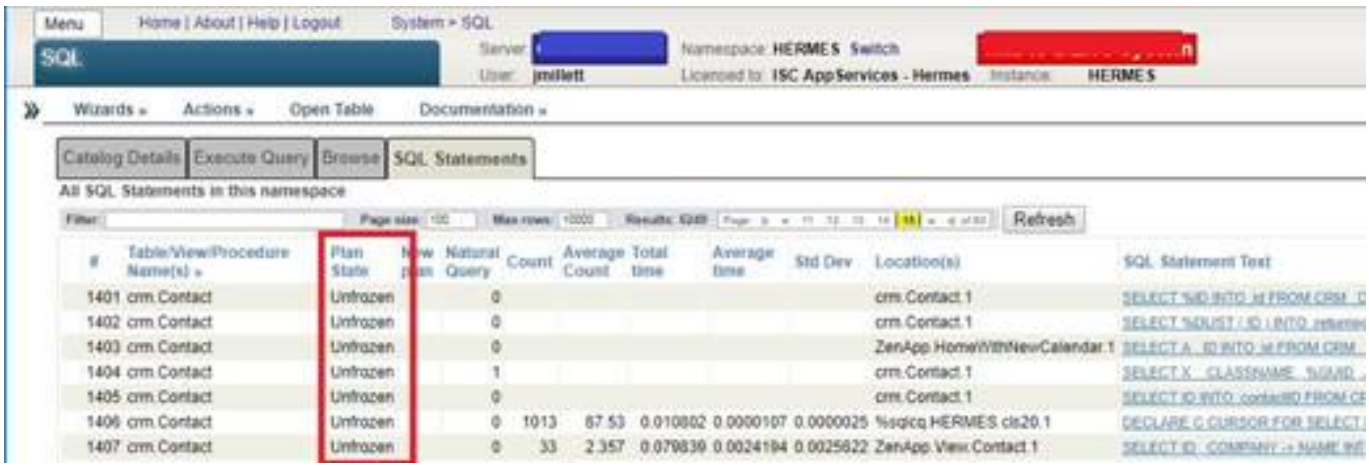
We saw this improvement on our development system and our test system. However, when we moved the code to the production system, the query still took “forever”. What went wrong?

- We checked that the code was properly moved to the live system.
- We checked that the code was properly compiled.
- We checked that the deployment script that built the new index ran successfully.
- We looked in the globals to double-check that the index was indeed properly built and populated.

However, when we looked at the output of SYS.MONLBL (a “heavy-duty” performance analysis tool that determines how often lines of code are being run), we saw that the query was still doing a complete scan of a large table for each item the code needed to check. After consulting with various colleagues (“Hat-tip” to @Timothy Leavitt and @Matthew Giesmann for great input and to @CarlFroneberger for generating the performance analysis data), we confirmed that the SYS.MONLBL output showed the query didn’t use the new index.

Why not? The query plans for the system, including the plan for the query in question, were frozen! The frozen plan had no idea about the new index.

We confirmed (and later changed) the “frozen” state of query plans using the Management Portal (System->SQL->”SQL Statements”). Part of such a display is shown here; note the “Plan State” column now shows query plans as “Unfrozen” as a result of Step 1 further below:



Why were the plans frozen? We had recently done an upgrade on the systems. Beginning with version 2016.2.0, Ensemble freezes query plans as part of “major” upgrades (many users don’t want upgrades

to affect query plans that they had previously finely-tuned). (An excellent webinar by @Kyle.Baxter describes why Ensemble/IRIS freezes plans on upgrade: https://learning.intersystems.com/course/view.php?id=969).

In our situation, however, we did want new query plans so that the new index would be used. We performed the following steps to make that happen:

1. d $System.SQL.FreezePlans(0,1)
2. Recompile the class that had the new index.

Note that the 1st parameter (“0”) directs the $System.SQL.FreezePlans method to “unfreeze the query plans” and the 2nd parameter (“1”) tells it to do so in the current namespace. More information about this method can be found in the class documentation for “%SYSTEM.SQL. https://cedocs.intersystems.com/latest/csp/documatic/%25CSP.Documatic.cls

More general information about Frozen Plans can be found in the following section of the “Caché SQL Optimization Guide”: https://cedocs.intersystems.com/latest/csp/docbook/DocBook.UI.Page.cls?KEY=GSQLOPT_frozenplans

#Indexing #Performance #SQL #Caché #InterSystems IRIS

Source URL: https://community.intersystems.com/post/case-thawing-frozen-query-plans-after-upgrade