<http://webcache.googleusercontent.com/search?q=cache:4VNxWj_XaYoJ:robertsdb2blog.blogspot.com/2010/12/db2-for-zos-keycard-gets-its-due.html+&cd=1&hl=en&ct=clnk&gl=in>

DB2 for z/OS: KEYCARD Gets its Due

By now, you've probably seen and/or heard a good deal of information about DB2 10 for z/OS, which was announced and became generally available this past October. There is indeed a lot of big news associated with this latest release of IBM's mainframe relational database management system: reduced CPU costs, support for temporal data (tables with system and/or business time dimensions), a huge increase in the number of threads that can be concurrently active, a migration path to universal tablespaces, SQL Procedure Language enhancements (better performance, plus the ability to write user-defined functions using SQLPL), table-based data access control policies, and lots more (and I do mean "lots" -- check out the DB2 10 "What's New" manual). Amongst all the "wow" features of DB2 10 are some nice little nuggets of functionality that will positively impact system performance and administration. One of these -- and the subject of this blog entry -- concerns the KEYCARD option of the RUNSTATS utility.

Here's the story in a nutshell: with DB2 10, the KEYCARD option is no longer optional. That is to say, if you run the RUNSTATS utility with an INDEX specification in a DB2 10 environment, and you have indexes with more than two key columns defined on the table (or tables) in the target tablespace, you will get the catalog statistics associated with KEYCARD, regardless of what you specify on the utility control statement. You can still put KEYCARD in this statement, but the keyword will be ignored because it "is now built into the normal execution of the RUNSTATS INDEX utility and cannot be disabled." Trust me: this is a good thing. It's a change that leaders on the optimizer team in IBM's DB2 development organization have lobbied for for quite some time. I'll explain why, and I'll tell you what this means for you if you're not yet running DB2 10 (and that's most of you, as, again, DB2 10 has only been generally available for a couple of months).

First, a little background. KEYCARD (which is only valid in the context of an INDEX specification on a RUNSTATS utility control statement) goes way back. I'm not sure when it was introduced as a RUNSTATS option, but I think that it might have been delivered with DB2 for z/OS Version 4 (mid-1990s). What does it do? Pretty simple: it causes RUNSTATS to determine the number of distinct values of combinations of an index's first "n" key columns, where "n" is greater than 1 and less than the total number of the index's key columns (I say this because the cardinality -- the number of distinct values -- of the index's first key column and of the full key are gathered anyway and placed in the FIRSTKEYCARDF and FULLKEYCARDF columns, respectively, of the SYSIBM.SYSINDEXES catalog table). The values obtained via the KEYCARD specification are placed in the SYSIBM.SYSCOLDIST table.

An example might be helpful here. Suppose you have a 10,000-row table containing data about customers in a particular country (one row per customer), and on that table you have an index with a key comprised of the columns STATE, CITY, and POSTAL\_CODE. Suppose further that you have customers in fifty cities, with each city being in a different state (i.e., one city in each of 50 different states -- admittedly, this is a rather contrived example). Finally, assume that there are 200 different zip code values in the table, and that the duplicate values of STATE, CITY, and POSTAL\_CODE are evenly spread across the table's rows (for information on non-uniform distribution of non-unique column values, check out an entry I posted last year on that topic in my old Catterall Consulting blog). Without KEYCARD specified, an execution of RUNSTATS will generate (as previously mentioned) FIRSTKEYCARDF (50) and FULLKEYCARDF (200) statistics for the index. With KEYCARD specified, RUNSTATS will also determine the number of distinct values of the index's first two columns: in this case, that's 50 -- the same as the FIRSTKEYCARDF value (if the index had 4 columns, KEYCARD would result in a determination of the number of distinct values of the combination of key columns 1, 2, and 3, as well as the number of distinct values of key columns 1 and 2).

Why is getting this statistical data important? Two words: column correlation. See, if you use what is a fairly common "base" RUNSTATS control statement, namely, TABLE(ALL) INDEX(ALL), DB2 will gather cardinality stats for every column of every table in the target tablespace. It will know, then, that there are 50 distinct values in the STATE column and 50 distinct values in the CITY column. What DB2 doesn't know is that the STATE and CITY values are highly correlated (CITY value 'Los Angeles' is paired with 'California', but not with any other value in the STATE column). Not knowing this, DB2 assumes that the values in the CITY column are independent of STATE values. Consider a query with the compound predicate below:

WHERE STATE = 'California'

AND CITY = 'Los Angeles'

Absent column correlation data, DB2 will estimate the number of qualifying rows by multiplying the cardinality of the STATE column by the cardinality of the CITY column; so, the estimate will be (1/50) \* (1/50) \* 10,000 rows = 4 rows. In fact, because STATE and CITY values are very highly correlated, the number of qualifying rows will be (1/50) \* 10,000 = 200 (again, I'm assuming a uniform distribution of duplicate STATE and CITY values in the table's rows). Lack of correlation stats for the STATE and CITY columns results in a filtering estimate that is off by a factor of 50. When a filtering estimate is way off from reality, the access path chosen by the DB2 optimizer for a query could be sub-optimal, and that path may cause the query to run a lot longer than it needs to. KEYCARD gives DB2 more column correlation information to use in estimating the number of rows qualified by a query's predicates, and better information leads to better access path choices (and better query performance, which of course is the bottom line).

So, why not have RUNSTATS gather correlation stats for all combinations of all columns in a table? That could be done, using COLGROUP specifications for groups of non-indexed columns, but the CPU and run-time costs of doing this would be very high unless the table had very few columns and not many rows (plus, coding all the COLGROUP specifications for a table with a lot of columns would be extremely tedious). In most cases, it just isn't a practical option. Making KEYCARD automatic in DB2 10 is a good idea because it generates data that has significant query optimization value (additional column correlation information for indexes with keys comprised of more than two columns) and does so at a low cost in terms of RUNSTATS CPU and elapsed time (this because the column groups for which correlation data is obtained are "leading and contiguous" subsets of multi-column index keys, making the correlation data cheaply available by way of a table's indexes, in which key values are always strictly ordered).

I'll conclude with a word to the wise: don't wait for DB2 10 to make KEYCARD automatic in your environment. Go ahead and add this option to your RUNSTATS INDEX specifications. If, for example, your basic RUNSTATS control statement includes

TABLE(ALL) INDEX(ALL)

I'd recommend changing that to

TABLE(ALL) INDEX(ALL) KEYCARD

You'll enrich your catalog stats at a low incremental cost, and you'll have taken a step towards smoothing the path to DB2 10, as you'll have in your catalog the correlation stats that will be automatically generated in the DB2 10 environment.

Troys article :

Table Sample Option Reduces RUNSTATS Overhead

January 29, 2013

The RUNSTATS utility gathers summary information about the characteristics of data in table spaces, indexes and partitions. I've previously written about using RUNSTATS (here and here), but now I can report that a new option makes the utility less resource-intensive.

It's no secret that RUNSTATS, in the course of conducting its valuable work, consumes a significant amount of CPU. However, with DB2 10, IBM has delivered a new option that allows database adminstrators to more closely control how RUNSTATS functions.

This summary information in RUNSTATS is used by the DB2 optimizer during the bind process to determine the best possible access path. Prior to DB2 10, the SAMPLE parameter was used to reduce the number of rows the CPU intensive cardinality calculation was performed on. However, 100 percent of these rows are still read by RUNSTATS. But now a new parameter, TABLESAMPLE, allows DBAs to determine the number of data pages that get sampled. Obviously, if only some pages are read, this can significantly reduce CPU usage.

So what percentage of pages need to be processed to ensure that the optimizer can determine the best access path? TABLESAMPLE includes a setting, SYSTEM AUTO, that allows DB2 to set the percentage anywhere from 100 down to 10 percent, based on table size. The larger the table, the smaller the sampling rate. For tables with fewer than 500,000 rows, all pages are sampled.

Alternatively, the percentage can be set manually. You can enter a value as low as 1 percent (0.01), though IBM doesn't recommend going below 10 percent.

The TABLESAMPLE parameter only applies to single-table table spaces, and is not applicable to LOB table spaces. Indexes do not exploit this page and row level sampling.

If you've used TABLESAMPLE, please leave a comment about your experiences.

Willies article:

Today we are going to focus on what DB2 11 has done to/for RUNSTATS...

Have you ever reached that point where you “think” the statistics in the catalog for a table, table space, or index are no longer correct/accurate? Maybe RUNSTATS keeps getting run with different control card options for the same objects or perhaps some of the stats were manually updated by someone who maybe wasn’t the best at knowing how to update the catalog statistics for an object (there are some special precautions you do have to take to make sure the correct associated set of stats are update).

Then there’s that dynamic statement cache. What do you do when you want to clear the statements for a certain set of objects in the statement cache because the access path they’re using is based on statistics you deemed are no longer correct (or accurate, or what you expect, etc).

DB2 11 has introduced a new option for RUNSTATS that makes the cleanup of everything above a very simple and straightforward task.

RESET ACCESSPATH

Just code up a “RUNSTATS TABLESPACE db.tsp TABLE tbl RESET ACCESSPATH” control card for RUNSTATS to reset the table and index access path statistics for a certain table space (or table spaces) back to their defaults. No statistics are gathers with this combination of options. It is only used to reset the access path statistics in the catalog. Once reset, the previous statistics values in the catalog for this object or set of objects is gone.

RESET ACCESSPATH will also invalidate and statements in the dynamic statement cache using the specified objects.

This option has NO affect on real time statistics.

If the option combination RESET ACCESSPATH HISTORY ACCESSPATH is specified, not only are statistics reset for the specified objects, rows are inserted into the catalog history tables SYSIBM.SYSTABLES\_HIST and SYSIBM.SYSINDEXES\_HIST recording when the access path statistics were reset.

Last comment about RESET ACCESSPATH; this option can only be specified with RUNSTATS TABELSPACE TABLESPACE. It is not valid with RUNSTATS INDEX.

Also, if your intent is to just clean-up old statistics history, your best bet still is to use the MODIFY STATISTICS utility.

However, be careful...

Don’t get the above confused with another stand alone RUNSTATS option called HISTORY ALL/ACCESSPATH/SPACE/NONE which controls the recording statistics in the catalog history tables. This option has been around for a while, nothing new and nothing changed here. This is not only a RUNSTATS option, it can also be specified in-line by the LOAD, REORG, or REBUILD. The default for HISTORY is also still controlled via the subsystem parameter (DSNZPARM keyword) STATHIST on the DSN6SPRM macro. The default if macro is not changed is NONE.

What is the most common cause of RID pool process failure?

Technote (FAQ)

Question

What is the most common cause of RID pool process failure?

Answer

The most common cause of RID pool process failures in DB2® in the z/OS® and OS/390® environments is not running RUNSTATS.

Many users are experiencing RID process failures. These failures can translate into nasty degradations such as table space scans and can be detected by the end user.

When enabling a list prefetch operation, the Optimizer sets a threshold that, if exceeded, will degrade the list prefetch operation into a table space scan. This threshold is currently 25% of the number of rows in the table. If RUNSTATS isn't run, the Optimizer assumes that the table contains 10,000 rows (the old default) and sets the threshold at 2,500. With the enormous growth in table sizes, it is very common for a list prefetch operation to exceed 2,500 entries scanned.