

OPTIMIZING DB2:

Get the Distribution Straight

BY RALF BERTLING

DB2's optimizer generally does

a good job of achieving the best possible performance for your SQL applications. However, normal RUNSTATS don't always provide the data necessary to choose the best access path. DB2 Version 8 uses column distribution >

statistics effectively, as Tom Moulder explained in his August/September *z/Journal* article, “The Most Important Feature in DB2 Version 8.” This article will go into greater detail about the difficult trade-off between RUNSTATS costs and their benefits, as well as the pitfalls in data retrieval and maintenance. You should consider getting distribution statistics, but be sure to keep them tidy and useful.

Column Distribution Statistics

In database tables, some value combinations occur more frequently than others. Column distribution statistics are a means of quantifying this kind of “data skew” as the frequency of certain value combinations. In DB2 V8, you can get this information by specifying the COLGROUP option for RUNSTATS, whereas older versions only allowed FREQVAL specifications (for leading column groups of an index) or the use of

such as CITY = :hostvar-city, the predicate STATE = :hostvar-state won’t exclude many columns.

Figure 1 shows the SQL I found on a customer’s production system (slightly modified for anonymity). Figure 2 shows the corresponding EXPLAIN output before and after applying column distribution statistics. In this example, you can see data skew on single columns. While HIGH2KEY could address the special TERM_DATE for a customer account from normal RUNSTATS, the optimizer would still assume that ENTRY_TYPE ‘B’ occurs as often as any other one and plan the access path accordingly. Since a nested loop join is efficient only on small result sets, the access path is changed to a merge scan join that has better characteristics on bigger result sets. The choice of the outer table may or may not be a consequence of the access method, but is less significant with

in all cases. (You typically change indexes via a DROP followed by a CREATE statement containing the refined design.) To be sure your statistics remain up-to-date, verify the content of your SYSCOLDIST table after index changes when you use RUNSTATS with FREQVAL.

Selecting Statistics

A problem with COLGROUP statistics is that the number of possible column combinations is close to 2^{COLCOUNT} , so it’s impossible to get statistics on every one of them. Ideally, you’d occasionally trace your SQL, parse it for JOINS, then sort and accumulate data based on affected columns and their performance to build lists of candidates for correlation statistics. As FREQVAL statistics need fewer resources, these might be used as an additional indicator to find out if the actual COLGROUP you’re interested in is significantly skewed.

It might be laborious to use tracing and indicators unless you can automate this process or buy a tool to do so. However, there is a simple approach that should be sufficient in many situations. As an easy way to get candidates, you can also look at your SQL statements, either by code inspection or by searching DB2’s plans, packages, and the dynamic statement cache. JOINS are generally interesting and unless you’re already deploying other tools, you should try IBM’s Visual Explain and use its statistics advisor component to determine which statistics might be beneficial. Since the COLGROUP statistics are expensive and data might turn out to be significantly skewed, you’ll need to plan the statistics execution and maintenance.

Maintaining Special Statistics

Without changes in your data model, the data skew will typically only change slowly, so it would be wasteful to include COLGROUP specifications on every RUNSTATS. It’s also important to drop statistics that don’t change the access path, as they’ll slow down BINDs and executions of dynamic SQL by the extra search time on the CATALOG, and you won’t need to regularly update COLGROUP statistics for that group. (You can use EXPLAIN before and after creating the statistics to find out if they do.)

To prevent trying the same candidates repeatedly, some sort of blacklist is advisable. In contrast, you might want to use some indicators for candidates (or re-evaluation) that will help you

When you’re binding or preparing dynamic SQL, the DB2 optimizer figures out an access path based on the estimated costs of different access methods.

the DSTATS. DSTATS has been available since DB2 V5 and is available as a free download from IBM at <ftp://www.redbooks.ibm.com/redbooks/dstats/>.

When you’re binding or preparing dynamic SQL, the DB2 optimizer figures out an access path based on the estimated costs of different access methods. For single table SELECTs, choices are limited, and though they may be wrong due to skewed statistical data, the extra costs are usually tolerable. When multiple tables are JOINed, data skew may cause serious performance degradation because the optimizer might choose the wrong outer table or join method for the size of the intermediate result set. This happens because the optimizer makes assumptions about the filter factor, which is the proportion of data returned from the evaluation of a predicate. It assumes, for example, the filter factor of an EQUALS predicate would be $1/\text{CARDF}$.

For correlated columns and skewed data, most predicates have a lower selectiveness; they return more rows than DB2 would expect at that stage of the execution. Typical examples of correlated data are CITY, STATE and ZIPCODE, or FIRSTNAME and GENDER in a customer table. After applying a predicate

merge scan joins anyway. Further examples and explanations can be found in Tom Moulder’s article.

Deleting Obsolete Statistics

No matter how you gather your distribution statistics, you should know how to get rid of them. Because column correlation statistics are comparatively expensive to create, you might not want to apply your normal maintenance cycle to them. That’s probably why IBM chose not to delete old column distribution statistics when you perform a RUNSTATS *without* special options. To remove statistics, specify the FREQVAL or COLGROUP options with a COUNT 0 or just delete the obsolete data from SYSCOLDIST. Since it only needs to scan index data in sorted order, RUNSTATS will be faster with the FREQVAL specifications than with the COLGROUP statistics.

When using the COUNT 0 method described above, you should take additional measures after dropping or changing indexes because SYSCOLDIST will still hold data related to the columns that used to be leading index columns. So you might end up with outdated statistics, since those entries aren’t automatically updated or deleted

avoid expensive COLGROUP RUNSTATS that are likely to return a result similar to the statistics in the catalog. Examples for such a test would be issuing a (cheaper) FREQVAL specification for a subset of the desired columns and checking the statistics for implausible data (e.g., SUM(FREQUENCY) > 1,

COLCARD(C1)* ... * COLCARD(Cn) < GROUPCARD(C1, ..., Cn), CARDF(table) < CARDF(column) or < CARDF(COLGROUP), or comparing the frequencies for concrete entries by issuing SELECT COUNT * SQL).

Ideally, you'd want to have your own bookkeeping about your candidates,

including tables with the group's status that can be toggled around (see Figure 3). The blacklist can also be used in earlier stages of candidate election, depending on your method.

As shown in Figure 3, you should collect candidates from various sources as desired. Remove candidates already blacklisted or processed from the list before issuing an EXPLAIN-RUNSTATS COLGROUP-EXPLAIN processing chain. Verify the access paths and keep or delete the statistics accordingly while saving bad candidates on the blacklist. Occasionally, use indicators such as time passed, RTS values, and the sanity checks mentioned above to repeat the verification process. The collection of candidates should be repeated as needed (depending on the method).

Conclusion

Column correlation statistics aid the optimizer in finding good access paths for JOINS, even when the underlying data is skewed. If the COLGROUP matches leading columns of an index, it's advisable to use FREQVAL instead of COLGROUP, but be sure not to screw up your index design to save on the RUNSTATS. Occasionally, you should check the DB2 catalog for outdated, implausible or useless statistics and update or delete them.

Remember, you probably don't need distribution statistics for every table. Whatever method you use to find your candidates, make sure you remove all statistics that don't cause your access path to change. This way you'll avoid polluting the catalog and get the maximum benefit from column distribution statistics while avoiding the pitfalls.

Feel free to contact me if you have questions about these concepts; I'll try to respond quickly. **Z**

```

DECLARE CURS_ABC CURSOR FOR
SELECT  A.A_COLUMN FROM CREA.MYTABA A, CREA.MYTABB B
WHERE   A.CUNO    = :CUNO   AND   A.CUNO = B.CUNO
      AND A.ENTRY_NO = B.ENTRY_NO
      AND A.TERM_DATE = '9999-12-31' true for 97.982938% of the rows
      AND A.ENTRY_TYPE = 'B'       true for 99.389265% of the rows
      AND A.ENTRY_TYPE = B.ENTRY_TYPE
      AND B.MODIFICATION_TS IN
      ( SELECT  MAX(B.MODIFICATION_TS)
        FROM    CREA.MYTABC
        WHERE   B.CUNO = A.CUNO
          AND   B.ENTRY_TYPE = 'B') true for 99.116081% of the rows
FOR FETCH ONLY

```

Figure 1: An Example of a Problematic JOIN

Before:

| LINE | QNO | PNO | SQ | M | TABLE_NAME | A | CS | INDEX | IO | UJOG | UJOGC | P | CE | TYPE |
|-------|-----|-----|----|---|------------|----|----|---------|----|------|-------|---|----|--------|
| 28567 | 01 | 01 | 00 | 0 | MYTABA | I | 01 | F943PI0 | N | ---- | ---- | | | SELECT |
| 28567 | 01 | 02 | 00 | 1 | MYTABB | I | 02 | F91EPI0 | N | ---- | ---- | | | SELECT |
| 28567 | 02 | 01 | 00 | 0 | MYTABC | I1 | 02 | F91EPI1 | Y | ---- | ---- | | | CORSUB |

After:

| LINE | QNO | PNO | SQ | M | TABLE_NAME | A | CS | INDEX | IO | UJOG | UJOGC | P | CE | TYPE |
|-------|-----|-----|----|---|------------|----|----|---------|----|------|-------|---|----|--------|
| 28567 | 01 | 01 | 00 | 0 | MYTABB | I | 01 | F91EPI0 | N | ---- | ---- | | | SELECT |
| 28567 | 01 | 02 | 00 | 2 | MYTABA | I | 01 | F943PI0 | N | -Y-- | ---- | L | | SELECT |
| 28567 | 02 | 01 | 00 | 0 | MYTABC | I1 | 02 | F91EPI1 | Y | ---- | ---- | | | CORSUB |

Figure 2: EXPLAIN Output Corresponding to Figure 1
Note the difference in access order and method.

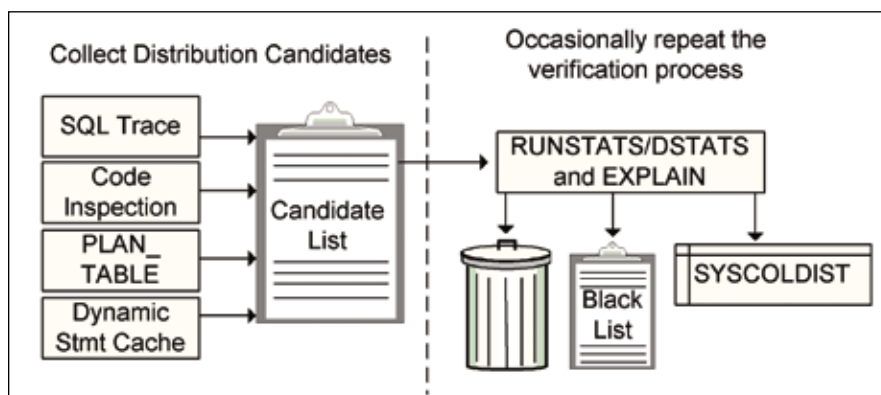


Figure 3: Workflow for RUNSTATS COLGROUP Candidate Selection and Maintenance

About the Author



After obtaining his master's degree in Computer Science, Ralf Bertling spent two years in university research before moving to the DB2 mainframe world as a developer. Currently, he's with SOFTWARE ENGINEERING GmbH,

a company focused on DB2 for z/OS maintenance, performance, and administration tools.

e-Mail: r.bertling@seg.de

Website: www.seg.de/