Filter factor statistics collection strategies

1.0 Introduction

The focus of this paper is to discuss statistics collection strategies for statistics used in filter factor estimation for columns which are more expensive and/or more difficult to collect. Different customers have different priorities regarding statistics collection and refresh strategies. Some people want to minimize the amount of analysis to be performed by a person. Some people want to minimize the amount of machine resources consumed in the collection of statistics. This paper provides information which should be useful for customers that would like to develop a customized statistics collection and refresh strategy to collect statistics which provide maximum benefit, minimize risk of stale statistics, and minimize the cost of statistics refresh.

1.1 Why statistics are required

DB2 for z/OS query optimizer is a cost based optimizer. The optimizer first performs a series of query transformations to increase the number of access paths available. After query transformation, the optimizer estimates the cost of the candidate access paths and selects the path with the cheapest estimated cost. The cornerstone of cost based optimization is a robust set of statistics which allow the optimizer the opportunity to accurately estimate candidate access path costs and differentiate efficient access paths from inefficient ones.

A common problem with costing SQL statements is obtaining sufficient statistics to accurately determine an efficient access path. It is possible to obtain an efficient access path with insufficient and/or inaccurate statistics, but it is typically less likely. Access paths based on insufficient or inaccurate statistics tend to be more unstable (e.g. more likely to regress in the future). Usually more complete and accurate the statistics allow the optimizer to more accurately estimate the cost of the available access paths and identify the optimal access path. Query access paths which are based on inaccurate or insufficient statistics are more likely to regress due to inaccurate cost estimates of the candidate access paths.

One of the most common causes of regressions is the optimizer does not accurately estimate the cost of the available access paths, and by chance selecting an efficient access path anyway. Sometimes the optimizer selects an efficient access path with insufficient statistics due to deficiency in the optimizer from considering alternatives. Some change is then introduced (runstats, CPU upgrade, optimizer change) which causes differences in the cost of the available paths or new paths become available due to optimizer enhancement. The optimizer selects the access path with the cheapest estimated cost, which in reality performs worse than before. Often the root cause of this problem is the statistics available are inaccurate or insufficient so the optimizer cannot differentiate between efficient and inefficient candidate plans.

In the case of optimizer enhancement where the optimizer used to select an efficient access path but now does not, the optimizer may never have differentiated between efficient and inefficient access paths. Often the optimizer simply has more alternatives than were available before. The new optimizer options offer both opportunity for improvement in performance and opportunity for regression. The key to obtaining the performance improvement rather than regression is the

optimizer may require more accurate and complete statistics to select the most efficient access path.

2.0 Categorizing filter factor statistics

The focus of this paper is on collection of statistics which are more expensive or difficult to collect. It is still important to ensure that statistics which are inexpensive and easy to collect are collected and refreshed to ensure they are accurate. I believe there are many missed opportunities for providing optimizer accurate filter factor estimates while minimizing risk and machine resource consumption in the easy / expensive, and harder / more expensive categories.

2.1 Easy, inexpensive

Table statistics (number of rows, number of pages, compression percentage)
Index statistics (index cardinalities, clusterratiof, number of leafs, number of levels)
Column statistics on leading indexed columns (cardinality, high2key / low2key, frequency)
Concatenated column statistics on leading indexed columns (cardinality, frequency)

2.2 Easy, more expensive

Some column statistics on non-leading indexed columns, non-indexed columns (cardinality, high2key / low2key)

2.3 Harder, more expensive

Column frequency statistics for non-leading indexed columns, and non-indexed columns.
Concatenated column cardinality on non-leading indexed columns, non-indexed columns
Concatenated column frequency

2.4 How to collect cardinality and frequency statistics

For information on how to collect filter factor statistics, visit DB2 for OS/390 and z/OS support web site and search for "Access path statistics." One of the links should be "How to collect cardinality and frequency statistics."

3.0 Statistics collection

In this section, I define what I consider to be base statistics. In my experience when the base statistics are available and accurate, the optimizer typically selects an efficient and stable access path. The optimizer only requires column / column group statistics for column / column groups used as where clause predicates in SQL statements. By identifying those columns used as predicates, you can limit the columns for which statistics should be collected.

3.1 Base statistics:

- Statistics on every table
- Statistics on every index with KEYCARD
- Single column cardinality on all columns used in where clause
- Multi-column cardinality on all multi-column join columns
    - Join columns which are leading indexed columns obtain multi-column cardinality via KEYCARD on runstats and/or FULLKEYCARDF collected on indexes.

- For multi-column joins where the join columns are not leading concatenated indexed columns it is desirable to collect this statistic to provide an accurate bound for the join size estimation.
- Single column frequencies on columns with data skew

In my experience, most queries which have the suggested base statistics obtain a sufficiently accurate cost estimate to select an efficient access path. For queries which use markers (host variables, parameter markers, or special registers) and the column associated with the marker has skewed data distribution, REOPT(VARS) is necessary to allow the optimizer to consider the data skew.

3.2 More advanced statistics

It is possible queries will require additional statistics including multi-column cardinality and multi-column frequency statistics to sufficiently differentiate efficient from inefficient access paths but this occurs less often. I suggest collection of additional multi-column cardinality and multi-column frequency statistics only when you are unable to obtain an efficient access path and when users identify correlation between non-leading indexed / non-indexed columns, and/or multi-column skew.

4.0 Statistics refresh

Databases often contain data which fluctuates over time. As data changes over time, it is necessary to refresh statistics to continue to provide the optimizer information which accurately reflects the state of the data. If the optimizer does not have accurate information the optimizer is more likely to select an inefficient access path due to inaccurate cost estimates. There is less value for your analysis time in customizing a strategy to collect statistics categorized as easy and inexpensive, so I suggest these statistics be refreshed regularly.

With an understanding of queries, frequently used predicates / predicate combinations, and data volatility, a statistics refresh strategy could be developed which maximizes the benefit these statistics provide while minimizing the cost of refreshing the statistics and the risk of the statistics becoming stale. Statistics become stale when the information they provide no longer accurately reflects the state of your data. The optimizer only requires column / column group statistics for column / column groups used as where clause predicates in SQL statements. By identifying those columns used as predicates, you can limit the columns for which statistics should be refreshed. Let s'observe the different ways in which a statistic could become stale.

4.1 Single column statistics

4.1.1 COLCARDF

Column cardinality is the number distinct values for the column data. Column cardinality is used for all types of predicates. So any column used as a where clause predicate should have column cardinality collected and refreshed based on volatility.

Some columns have cardinality growth, and some columns have static cardinality. The cardinality of primary key columns for example would grow as the table grows - because they

are unique. Other columns which tend to incur column cardinality growth are columns like last_name, first_name, city, state. The cardinality of these columns may grow rapidly initially as a company grows - the columns will have new names which have not yet existed, and they will have new cities and states. Over time the rate of growth of the cardinalities may slow, but there still might be some growth. Some columns have little or no column cardinality growth. Gender is one example. Gender likely has column cardinality of 2 and never changes. Often customers have indicator flags where there are a limited number of possible values.

Columns which have volatile cardinality will periodically require statistics be refreshed to reflect the cardinality growth. Columns with static column cardinalities would not benefit from refreshing statistics.

4.1.2 HIGH2KEY / LOW2KEY

HIGH2KEY is the second highest data value for the column. LOW2KEY is the second lowest data value for the column. The purpose of HIGH2KEY and LOW2KEY is to identify the range of typical values for the column. The second highest and second least values are used rather than highest and lowest values because often the highest and lowest values are used as defaults and do not accurately reflect the range of typical values for the column.

HIGH2KEY/LOW2KEY are primarily used for linear interpolation of range predicates. Linear interpolation is only possible when there is a ***COL range literal*** predicate. For example: WHERE BIRTH_DATE BETWEEN '2002-12-01' AND '2002-12-31' use linear interpolation because the literal values are known. If parameter markers, host variables, or special registers are used (e.g.WHERE BIRTH_DATE BETWEEN ? AND ?) then linear interpolation is not used because linear interpolation requires the literal value to compare the range searched over the entire range. So HIGH2KEY / LOW2KEY are only used when there is a predicate with COL range LITERAL.

If the range of values changes then HIGH2KEY / LOW2KEY should be refreshed. For example some customers populate and purge data based on date columns. Presume there is a rolling 1 year range of data in a table. Currently the table contains transactions with dates between '2001-12-01' to '2002-12-01'. A month goes by and data with TX_DATE < '2002-12-31' is purged and data for December 2002 is inserted. If statistics are not refreshed and a query contains a predicate such as WHERE TX_DATE BETWEEN '2002-12-01' AND '2002-12-31' is executed, the SQL may perform poorly.
This is because HIGH2KEY is stale - near value 2002-12-01 and the range being searched for is outside of this range. The optimizer will presume that very few rows qualify for this predicate. This may result in inefficient access path being selected. So in cases where the range of values changes and the column is used as a COL range LITERAL predicate, the column statistics should be refreshed to provide optimizer an accurate range.

Some astute customers have recognized when even HIGH2KEY or LOW2KEY do not reflect typical values, and manually adjusted HIGH2KEY and/or LOW2KEY to provide better reflect the range of most values. For example assume an automotive service center

keeps track of it s customers, and one of the columns is automobile MODEL_YEAR. For this service center, 95% of the customers have automobiles with a MODEL_YEAR between 1980 and today. There is a rare set of customers with older and sometimes classic cars which skews the low end of the range into the 1940 s. In this case the typical values for this column are between 1980 and today, so the optimizer may estimate more accurately if the LOW2KEY is increased to around 1980.

There is one situation where HIGH2KEY / LOW2KEY are used in absence of a range predicate. When the COLCARDF for the column is 1 or 2 and there are no frequencies collected on the column, the optimizer will use HIGH2KEY and LOW2KEY to manufacture frequency statistics.

Typically columns which are never used as range predicates and have a column cardinality greater than 2 do not require accurate HIGH2KEY / LOW2KEY. In these cases, HIGH2KEY / LOW2KEY is not used. There are also many columns such as the GENDER example, status, and indicator flags where the range of values does not change. It s really more important to identify those columns where col op range predicates **are** used and the range **does** change so a plan can be developed to refresh appropriately. DATE and TIMESTAMP columns are often used as col op range predicates and the range of data often changes. It is typical for queries to search for recent dates. Stale statistics in this situation case may result in inefficient access path.

4.1.3 FREQUENCY

Frequency statistics show the distribution of data on specific column values. Often some values of data occur much more frequently than other values. To provide the optimizer information about data skew, frequencies should be collected. Since frequencies are stored on specific values, the optimizer would require knowledge of the literal value to use frequency statistics. If markers are used to search on columns with skewed data distribution then REOPT(VARS) should be considered. There are several ways these statistics can become stale.

4.1.3.1 Distribution of data changes

Presume a table with column COUNTRY to reflect the country a customer lives in. Initially, most customers are from country Spain. But there are many customers with COUNTRY of France, Portugal, and UK. Then there is a very successful marketing campaign which results in many new customers in France. In this case, the distribution of data becomes more skewed towards France. The number of countries which have customers has not changed but the distribution of data across countries has changed. Columns of this nature should be refreshed.

4.1.3.2 Values in domain change

Let s use the TX_DATE column again. Presume there is a table which contains a rolling 1 year of transactions. The data is uniformly distributed across months (so each month has the same number of records as every other month). Currently the table contains data from 2001-12-01 to 2002-12-01 . At the end of December the data with TX_DATE <= 2001-12-31 is purged and data is inserted for dates between 2002-12-01 and

2002-12-31 . 'If frequencies are not refreshed it is possible there are frequencies which indicate significant portions of the data have values before date 2001-12-31 , when in reality now no rows exist. Since frequencies have not been refreshed, there will be no frequencies indicating there are rows with values > 2002-12-01 . 'If user wrote a query looking for 2002-12-01 through 2002-12-31 the optimizer likely will presume that very few rows exist based on the HIGH2KEY / LOW2KEY and the frequencies.

This issue is particularly important when frequencies are collected on all column values. If the number of frequencies is close to or equal to COLCARDF for the column, then the optimizer has statistics on the entire domain. In this situation, when the optimizer compares the range or value being searched for in the query with frequencies, the optimizer finds 100% of the data is values which you are not searching for. Stale statistics in this situation can result in significantly inaccurate filter factor estimate (no rows qualify) which certainly can result in inefficient access path. In this case frequencies should be refreshed. If the column values are volatile and the distribution of data is fairly uniform (as is the case here), then another solution is to not collect frequencies on this column. When data is uniformly distributed frequencies are of little or no benefit. If data is uniformly distributed with volatile column values, there is little benefit coupled with significant risk of regression if the statistics are allowed to become stale.

4.1.3.3 Distribution of data and values in domain change

Assume there is a table which tracks members of a recreation club and the activities the members are participating in. Currently services are provided for activities football, soccer, volleyball, swimming, and table tennis. The recreation center is doing well and members express an interest in more activities. Activities are added for diving, basketball, and hockey. No activities are eliminated. Based on new options, the number of people participating in all of the activities changes. Some of the members which only registered for football and soccer now only participate in basketball and hockey. Some members registered for multiple activities. Table tennis is dropped as an activity due to poor registration. At this point, some activities have been added, some activities dropped, and the distribution of the data according to all activities has changed.

For some columns the values and the distribution of data changes. In this situation there is a tradeoff. The statistics for this type of column should either be aggressively refreshed to ensure the statistics reflect reality or frequencies for this column should not be collected - avoiding the risk of the frequency statistics becoming stale. If queries require an accurate estimate of selectivity on this type of column to obtain efficient access paths, then the statistics should be collected and refreshed as needed. If the statistics are not necessary for efficient access path, then frequency statistics on this column should be avoided to eliminate risk of statistics providing inaccurate information.

Sometimes it is known when the data in a column is going to change, and the statistics refresh can be synchronized. For example, statistics could be refreshed after registration completes at the recreation center. The distribution of data on a columns like BUSINESS_UNIT may stay static until a business unit is purchased or sold. When fluctuations are predictable then

an appropriate collection strategy can be defined. This is a judgment call. Is the data sufficiently skewed that the optimizer requires frequencies to select an efficient path? If not, then frequencies should not be collected on this column. If so, then frequencies should be collected and refreshed periodically depending on the volatility of the data.

4.1.3.4 Neither distribution of data nor values in domain change

Often the frequencies which provide the most value are on columns where the statistics are fairly static - even as other statistics on the table change. For example assume a bank has a table with column DELINQUENT_LOAN which has Y/N as valid values. For this successful bank there are a small set of customers with delinquent loan = 'Y'. The marketing department is promoting customers for new credit card services and decides not to market customers with delinquent loans. The following predicate is added to the query WHERE LOAN_DELINQUENT = 'N'. Over 99% of users have LOAN_DELINQUENT = 'N' and less than 1% have LOAN_DELINQUENT = 'Y'. This is the type of column where there is not likely to be a very significant change in the distribution of data.

Columns which have skewed data and where neither the distribution nor the values change over time can have statistics collected once, then ignored for long periods of time - perhaps indefinitely. If the frequencies are not collected at all optimizer assumes uniform distribution so optimizer would estimate ½ the rows are returned when in reality either 99% qualify or 1% qualify. There often are several columns of this nature used as predicates in queries. Collecting frequency statistics on this column will provide the optimizer valuable data skew information with little need for refresh and little risk of the statistic becoming stale.

4.1.3.5 Static skewed distribution on default / null value

Often columns have a default value or NULL, the data is highly skewed on one (or both), and the distribution of the data on the default / null value is static (does not change much over time). The remainder of column values do change and/or the data distribution changes. One customer had a column 'DATE_OF_DEATH'. The default value for DATE_OF_DEATH is '9999-12-31'. As customers pass away, the date of death is changed. This column has a fairly high column cardinality because there are many days in which customers have passed away. The customer may wish to only promote customers which are alive, so predicate WHERE DATE_OF_DEATH = '9999-12-31' is often used.

It is common for a column to be skewed on a default / NULL value. Often the SQL which performs poorly is when the query searches for the default / NULL value. When a column has data skew on the default / NULL value consider collecting frequencies on just the default / null value(s), and not any other values. The optimizer will more accurately estimate filter factor when queries search for the default / NULL value as well as when queries search for other values when the value is known.

A query which has predicate WHERE DATE_OF_DEATH = '2002-1-12' will use the frequencies on default / NULL value, taking into account the remaining number of unknown values and that MOST of the table has value '9999-12-31'. The optimizer assumes the remainder of the table is uniformly distributed for values for which statistics have not been

collected. By collecting frequencies only on the default value and/or null there is lower risk of the frequencies becoming stale.

4.1.4 Single column summary

When determining what statistics need to be collected and refreshed, start with the base statistics identified early in the article. If there are concerns regarding the expense or risk of collecting and refreshing frequencies it may be useful to categorize columns according to 4.1.3.x sections, and initially only collect frequencies on columns categorized as **4.1.3.4 Neither distribution of data nor values in domain change** or **4.1.3.5 Static skewed distribution on default / null value.** Often by correcting filter factor estimation on this group of columns alone can result in efficient access path. In my experience predicates on these columns can have more significant filter factor error than other types of columns because the data skew can be so acute on default values and status flags. It is possible by obtaining a more accurate filter factor in just these cases you could obtain efficient access path. If these statistics are not enough, then collection of frequencies on other columns may be necessary, or alternative tuning techniques could be used.

4.2 Multi-column statistics

The multi-column statistics available are multi-column cardinalities and multi-column frequencies. I suggest refreshing multi-column cardinality and frequency statistics whenever any of the columns which are part of the column set require refreshing. If all of the columns in the column set are static, then the statistic could be collected infrequently. If any of the columns in the column set are volatile on a single column cardinality, then the multi-column cardinality should be refreshed at same time. If any of the columns in the column set are volatile on data distribution or on what values the data is distributed on, then the multi-column frequencies should be refreshed when the single column frequency is refreshed. As a volatile column changes, the cardinality, values, and data distribution on the entire column group may be affected.

5.0 Summary

The optimizer uses statistics to estimate the cost of candidate access paths. The optimizer is more likely to generate efficient and stable access paths when sufficient and accurate statistics are provided. The statistics required to identify efficient access paths depend on many factors including what predicates are used in the SQL statement, physical design of the database, and the candidate access paths considered by the optimizer. Inefficient access paths can be chosen due to many reasons but often the reason is insufficient or inaccurate statistics which results in accurate cost estimates for the candidate access paths. When a query obtains an inefficient access path or a query regresses my suggestion is to review the availability and accuracy of the base statistics identified earlier to determine if this may be the cause of the inefficient access path.

Some customers collect and refresh all the statistics which possibly may be required at the same interval. In some cases collection and refresh of all statistics is too resource intensive. With some analysis of queries to determine common predicates / predicate combinations, and analysis of the volatility of columns used as predicates a tailored statistics collection and refresh strategy could be devised which provides the optimizer sufficient information to select efficient access

paths, addresses the risk of stale statistics, and minimizing the cost and frequency of refreshing statistics.