



ACTIVATE BUSINESS WITH THE POWER OF I.T.™



## DB2 UDB for z/OS

SQL Performance Tuning

By Tom Moulder, President, TREX Associates, Inc.



**Associates, Inc.**

# Table of Contents

- ABSTRACT** ..... 1
- WHY TUNE SQL?** ..... 2
- SQL PERFORMANCE TUNING BASICS** ..... 2
- SQL PERFORMANCE TUNING VARIABLES** ..... 2
- THE IMPORTANCE OF DATA** ..... 2
  - > Execution Metrics ..... 2
  - > Explain Data ..... 3
  - > Object Statistical Data ..... 4
  - > Verifying the Input ..... 4
- TUNING OPTIONS** ..... 4
  - > Changing the SQL ..... 4
  - > Predicate Analysis ..... 5
  - > Writing to Available Indexes ..... 6
  - > Optimize Subqueries ..... 6
  - > Sequence of Table Access ..... 6
  - > Changing the Schema ..... 7
  - > Predicate Usage ..... 7
  - > Read vs. Update Frequencies ..... 7
  - > Correlated Columns ..... 7
  - > Impact on Maintenance ..... 7
  - > Column Values ..... 8
  - > DB2 Limits ..... 8
  - > Can't Change the SQL ..... 8
- USE A HINT** ..... 8
  - > Issues with Hints ..... 8
- CHANGING OR COLLECTING OBJECT STATISTICS** ..... 9
  - > Cardinality ..... 9
  - > Distribution ..... 9
  - > Correlation ..... 10
- OPTIMIZE SQL PERFORMANCE ACROSS THE APPLICATION LIFECYCLE** ..... 10
- KEY FEATURES AND BENEFITS** ..... 10
- CONCLUSION** ..... 11
- BMC SQL PERFORMANCE FOR DB2** ..... 11

## Abstract

This white paper was prepared by TREX Associates, Inc. for the exclusive use of BMC Software, Inc. and its customers to provide SQL tuning strategies for DB2 UDB for z/OS®.

Performance tuning of DB2 UDB for z/OS subsystems has been a common task since the announcement of that database subsystem by IBM. Performance tuning can be accomplished at the subsystem level or at the SQL statement level. It is generally accepted that 20% of the benefit from tuning can be accomplished at the subsystem level, and 80% of the benefit will be derived by tuning at the SQL statement level. This paper discusses performance tuning at the SQL statement level.

## Why Tune SQL?

IT organizations face significant challenges to deliver the computing services required by the modern global business environment. These challenges include meeting service level agreements that include stringent response time goals. This issue and business continuity are the two greatest issues facing businesses today. The move to make legacy applications available through web interfaces has added additional processes that increase response time. The pressures to maintain service have caused companies to make a choice between increasing computing capacity at significant cost and improving the performance of their business applications.

The cost of a DB2 application includes many factors. Businesses now realize that the time spent improving the performance of a DB2 application and the software to assist in that task is justified. This is a difficult task for the following reasons:

- > Lack of qualified personnel
- > Large number of variables to consider
- > Complexity of the task

This paper will provide a basis for evaluating the performance of existing and planned DB2 applications. Performance benchmarks continue to show DB2 UDB for z/OS is the most reliable and secure platform for mission-critical applications. Performance analysts should find this paper a valuable tool in the complex process of achieving maximum performance from DB2.

## SQL Performance Tuning Basics

Tuning SQL statements involves the analysis and prudent use of several factors depending upon the type of SQL and the performance tradeoffs that must be made. All performance tuning involves applying analytical processes to produce the most resource efficient means of accomplishing all the business tasks required by a business enterprise. Priorities are set based upon the importance of the line of business. Performance tradeoffs are made based upon the cost of resources to accomplish the required business functions. Information Technology has under its control the following resources:

- > CPU
- > I/O
- > Memory

When an SQL statement is made to perform better because an object is moved to its own buffer pool, a tradeoff has been made by using memory resources to reduce CPU resources. The tradeoffs vary based on the availability of these three resources.

There are hardware and software limits to these resources just as there are budgetary limits on the use of these resources. All limits must be taken into account.

## SQL Performance Tuning Variables

Several variables must be considered in tuning SQL. The type of SQL is a factor, so it is important to know if the SQL is static or dynamic. It is also important to know whether the SQL can be modified. This may seem like a strange variable to consider, but more and more purchased application packages contain DB2 SQL that cannot be changed without voiding any warranties and support contracts that may exist. There are still tuning options even when the SQL must remain the same. Consideration must also be given to the schema involved in the application and whether changes can be made to the schema. This would include such actions as adding or removing indexes within the schema. Another variable that is becoming more important is to know whether statistics have been collected on the application objects that are referenced by the SQL. Each of these factors play an important part in determining exactly what options are available for tuning and the priority of each option in the tuning process. This white paper provides a list of options for the tuning process to ensure that all options are considered as each SQL statement is tuned.

In addition to considering all the appropriate variables when tuning a single SQL statement, there must be an overall system perspective that determines the tuning opportunities based on potential return on investment of tuning time. This system perspective is important to ensure that performance tuning is carried forward on the highest priority applications, and on the applications that show the greatest opportunity for resource savings.

## The Importance of Data

Performance data is particularly important. While there is an art to tuning SQL statements, this art form is based upon concrete facts and figures that provide a basis for tuning recommendations, priorities, and accomplishments. SQL performance tuning data is derived from several sources:

- > Execution metrics
- > Explain data
- > Object statistical data

### Execution Metrics

An integral part of any performance tuning strategy is the collection of actual runtime statistics for as high a percentage of the total SQL as is practical from a budget and resource perspective. This execution information should have detailed data concerning the use of the three main resources that can be controlled – CPU, I/O and memory.

These metrics are collected by DB2 and made available through the Instrumentation Facility Interface. While the overhead of collecting high-level metrics is small, low-level metrics are more expensive and require evaluation before collection. One myth that exists concerning collection of these metrics is that multiple users of the data will cause additional CPU overhead for each user of the collected data. That myth is false, and any DB2 user should realize that the overhead of producing this data is incurred only once. After that is complete, the only additional overhead for these metrics is the cost of the storage media to maintain this data for whatever historical period is required.

CPU information should be as detailed as possible. This includes segregating the CPU usage into DB2 time and application processing time using Class 1, 2 and 3 times at a minimum. It is highly recommended to collect Class 7 and 8 times that further aggregate the CPU time used by plan and package. It would also include segregation of CPU time for calls to stored procedures. Any option to set thresholds based on CPU execution time is a valuable means to reducing the overhead of monitoring to fit within any budget constraints. Exception thresholds must allow enough granularities to set thresholds on statements that execute in less than 1 second CPU time. Milliseconds of execution CPU time can be significant for those in charge of performance tuning, based on the number of executions for a particular SQL statement.

I/O information should also be as detailed as possible. This would include aggregated counts and total elapsed timings for all I/O activities. Collecting this data for application objects by object name is expensive but very useful. Any time DB2 spends executing I/O processing for an SQL statement is time spent waiting for data. Because this time is better spent processing the data rather than retrieving the data, this information is important for reducing response time to meet and exceed service level agreements.

Memory information should also be as detailed as possible. DB2 applications can use memory in several ways. The two most prominent users of storage are the buffer pools for pages of application data and the EDM pool for application program and SQL statement data. This data is important to analyze when resources are constrained. Primarily this is a function of subsystem tuning and will only be dealt with in this paper as it relates to the tuning of SQL statements.

## Explain Data

DB2 provides a wealth of information documenting the input to the DB2 Optimizer that was used in formulating the chosen access path for retrieving the result set for each SQL statement. This information is stored in DB2 tables if they exist at the conclusion of the Bind/Rebind/Mini-Bind

processes. Some basic information concerning the DB2 Optimizer would be helpful at this time.

## DB2 Optimizer Basics

The importance of the DB2 Optimizer and its choices for access path selection cannot be over-emphasized. The DB2 optimizer will determine what it believes is the optimal means to access the required data for every SQL statement that executes. The Optimizer uses several important sources of input to perform this process:

- > Number and speed of available CPUs
- > Buffer pool configuration
- > Object statistics from the catalog
- > SQL statement text
- > Schema

The Optimizer uses many internal proprietary formulas to calculate an estimated cost for each query. Those formulas estimate the cost of CPU and I/O processing. As a result, the optimizer is sensitive to changes in processor speeds and the number of I/O operations required producing a result set.

When that access path selection process is performed is important.

## Static SQL Optimization

For Static SQL, the Optimizer will choose an access path at Bind/Rebind time. That access path is stored in the directory and is used for all subsequent executions of that SQL statement. At Bind/Rebind time, the user can also provide a value of YES to the EXPLAIN parameter and DB2 will externalize all the access path information to a set of tables commonly referred to as the Explain Tables. Creating and saving this information for a significant period of time allows for many comparison options in the tuning process.

## Dynamic SQL Optimization

For Dynamic SQL, the optimizer will choose an access path at execution time unless there is a dynamic statement cache. If there is a dynamic statement cache, then the access path will be stored with the statement in this cache. Subsequent executions of the same statement will use the same saved access path. The DB2 optimizer uses a similar set of control blocks for the calculation of the access path and the explanation of what the optimizer has chosen. Those control blocks are available after the access path selection process for a dynamic SQL statement and can be externalized to any valid IFCID destination through the use of IFCID 22. The information from this IFCID is not as complete as the Explain Tables from the Bind/Rebind process but is a good source of information for tuning purposes.

The SQL text data for dynamic SQL is available from DB2 through the IFCID 63. This data is valuable, but there must be a means of consolidating this data in such a manner that reduces the overhead of this collection. Many dynamic statements may have different literals specified for predicates. Some mechanism should be developed to treat similar SQL statements as equivalent. This requires parsing of the text and evaluating the different pieces of each statement to determine equivalency.

### DB2 Access path Choices

The following list shows the potential access paths for DB2 listed in the order of minimum to maximum resource usage when accessing objects with greater than 10 million rows in at least one table referenced in the statement:

- > Direct row access using ROWID
- > One fetch index scan using Min, MAX
- > Unique matching index scan using a predicate value
- > Matching index scan only
- > Non-matching index scan only
- > Matching index cluster scan
- > Matching random index scan
- > Multiple matching index scan using AND and OR
- > Non-matching cluster index scan
- > Segmented table space scan
- > Non-segmented table space scan (in parallel or sequential)
- > Non-matching random index scan

### Object Statistical Data

DB2 provides tables in the catalog to maintain statistical information about the data that resides in DB2 tables and indexes. This statistical information can be used for several purposes including:

- > Access path selection
- > Scheduling of reorganizations
- > Acquisition of storage to handle growth
- > Object placement

While all of these functions may be important to certain users, this paper examines the use of this statistical information by the Optimizer for access path selection purposes.

If this statistical information is not present (i.e., the values in these columns are -1), then the DB2 Optimizer will assume default values. The vast majority of the time, these default values will not generate an optimal access path selection for use by DB2 at execution time.

The statistics used by the Optimizer are listed the Version 8 Administration Guide (SC18-7413-03).

These statistical values allow the DB2 Optimizer to calculate an estimated value for I/O and CPU processing that is closer to reality than the estimates would be without these values.

The importance of these statistics and their presence in the DB2 catalog becomes more important for access path selection with every new version of DB2.

### Verifying the Input

All of the data that has been mentioned can be used to verify the accuracy of the data available to the Optimizer. The I/O counts from SQL execution can verify that row counts used by the Optimizer are close to reality. When wide variations in these values exist, there can be benefit from changes either in the input to the Optimizer or a Bind/Rebind to allow the Optimizer to choose a new access path for a particular statement. The benefit that can be derived from this process varies greatly and any automated process to quantify the benefit certainly provides great value. By quantifying the benefit, the changes that will provide the greatest benefit can be implemented first. Because justification for any change is helpful, if not required, any comparison of data provides input to the justification process and makes the performance tuning process more simplified. The justification process will also help the application users understand the priority associated with performance changes and how those changes compare to normal application changes to provide greater function.

### Tuning Options

When tuning SQL, four separate tuning opportunities can be considered. This paper discusses each of the available options and how each can be managed for optimum performance enhancements. The four options are:

- > Changing the SQL
- > Changing the schema
- > Use a hint
- > Changing or collecting object statistics

Let's analyze each of these options to determine the conditions under which each one would be viable.

### Changing the SQL

The first option for performance is to change the SQL.

This particular option is the most perplexing to discuss. If you have a purchased application package, you may not be able to change the SQL without great pain and agony. The vendors do not want you to change their SQL, and they go to great pains to make sure you don't. From the vendor perspective, it is much easier to support a product when you know that the underlying SQL has not changed. It might be possible to change SQL for any purchased product,

but the maintenance of those changes may be more costly than living with the poorly performing SQL or trying some of the other options.

The following describes the process of changing SQL to improve the performance of the statement. This description contains a series of conditions to examine for possible improvements in the SQL.

- > Predicate analysis
- > Writing to the available indexes
- > Optimize subqueries
- > Sequence of table access (multi-table SQL)

## Predicate Analysis

Predicate analysis provides an opportunity to address two concerns with the same data. Predicate analysis is a great place to start SQL tuning. Also, most of what is mentioned for predicate analysis will have some application to index design as well. Therefore, a great deal of time will be spent on this subject as it applies to two of the four tuning options.

Having good qualifying predicates is required to reduce the resource requirements of an SQL statement. Predicates should be evaluated for the factors that affect resource usage:

- > Whether the predicate is indexable
- > The predicate's processing stage within DB2
- > The order of predicate evaluation
- > Pruning and screening

### Whether the Predicate is Indexable

Whether the predicate is indexable is determined by a variety of factors. The simplest approach is to examine the Explain Tables that are populated by DB2 to determine if an available index for the predicate has been selected. The Administration Guide contains a table of all the rules that are applied by DB2 to determine if a predicate is indexable. You can apply these rules during development, or you can apply these rules when you find predicates in the Explain Tables that are marked non-indexable. Not all predicates in a query will be indexable. Knowledge of the application schema will help in determining when additional effort should be made to make a predicate indexable and when that effort would not be profitable.

### The Predicate's Processing Stage within DB2

DB2 has two stages for processing predicates numbered 1 and 2. Stage 1 processing occurs as pages are retrieved into the buffer manager. Elimination of non-qualifying rows at this point will reduce memory usage in the DB2 buffer pools and CPU processing of the rows during Stage 2 processing thus improving performance. Stage 2 processing occurs within the DB2 Relational Data Manager component and will result in additional CPU usage to perform any processing of

the query as well as increased memory in the buffer pools for all rows that are eliminated by Stage 2 processing. The Explain Tables will identify each predicate as being Stage 1 or 2. When you find predicates that are Stage 2, you should consult the Administration Guide for list of all the rules followed by DB2 to determine whether a predicate is Stage 1 or 2. Any change that does not materially affect the result set and changes predicate processing from Stage 2 to Stage 1 is a significant performance enhancement.

### The Order of Predicate Evaluation

The order of predicate evaluation is important as well. DB2 has three criteria for determining the order of predicate processing. Within each criterion, three rules are applied to determine the order within each criterion. Predicates are order for processing based on the following three criteria:

1. Indexable predicates
2. All other stage 1 predicates
3. Stage 2 predicates

With each of these criteria, the following three rules are applied to impose additional order upon predicate processing:

1. Equal predicates
2. Range and IS NOT NULL predicates
3. All other predicates

The order of predicate processing is important because this order allows DB2 to eliminate non-qualifying rows from the result set as soon as possible and avoid further CPU and I/O processing based on the non-qualifying rows. This overhead would be dependent upon the type of query and any other predicates. This will affect a join query because elimination of rows from an intermediate table will not be joined to subsequent tables depending upon the Join sequence. (More will be said about Join sequences later.) Within the same query, elimination of non-qualifying rows from one predicate may reduce the number of rows evaluated by future predicates. This will reduce CPU usage and possibly I/O processing.

DB2 Version 8 has made significant changes to the rules regarding equality predicates using parameter markers or host variables. In any mode of Version 8, DB2 will consider a predicate Stage 1 even when the column definition in the catalog is a different length from the parameter marker or the host variable. This should result in better access paths in Version 8.

### Predicate Screening and Pruning

Predicates that do not have an associated index or are not indexable can still be beneficial in reducing the resource consumption of a query. Any Stage 1 predicates that are not indexable are applied during Stage 1 processing after

all indexable predicates. Screening is only available for predicates with an Equal, Range or Null operator. Screening is used by DB2 whether the access path is a matching index or non-matching index access path. This process further reduces the number of qualified rows within Stage 1 rather than in the more CPU intensive Stage 2. This also means that further examination is required when comparing the output from the Explain command in DB2. Any screening predicates are not counted in the MATCHCOLS column of the PLAN\_TABLE, but they can have a significant impact on performance.

Additionally, any predicates that identify paths that would return no rows can be pruned to further reduce the number of access paths the Optimizer must evaluate. Pruning will reduce the resource associated with the Bind/Rebind/Mini-Bind processing. Several examples of Pruning predicates are:

- > Column <Operator> Literal or Host Variable\*
- > Between
- > IN (list)
- > AND'ed predicates
- > OR'ed predicates
- > Predicates "pushed down" from higher query blocks

\*Where <operator> is any equality operator such as =, <, >, <=, >=, <>.

Predicate analysis is an important process in tuning DB2 SQL statements. It is important to extract as much information as possible from the DB2 Explain process to reduce the man hours required in predicate analysis. The more the computer does for you, the faster this analysis can be accomplished.

### Writing to Available Indexes

Predicate analysis leads into the topic of indexes. There may be times when you find a predicate is not indexable because the column does not exist in any index. However, the process of adding an index is a complex task that should not be done without an analysis of the impact that index will have to the application as a whole as well as the one SQL statement that will benefit from the index. Consider writing predicates to take advantage of existing indexes where possible before considering adding additional indexes. Knowledge of the application schema will help determine whether other columns that do have indexes can provide the same result set.

When other columns with an index cannot be used for a particular query, then an analysis should be made to determine whether an additional index would improve the overall performance of the application. Information concerning predicate usage should be collected and viewed from an application perspective. For the column that you

consider indexing, would it be wise to make this column an additional key in an existing index or should this column have an index of its own? This can be determined by analyzing how often this column is used in the application and whether the use of this column is correlated to the use of other columns in the table. This predicate usage information is available from the index component of BMC APPTUNE and reflects actual SQL statement execution information as opposed to statements that exist in the catalog but may never have been executed. As with all execution data, you should be careful to understand the interval for the data that is being evaluated. Predicate usage information should span a long time period (one to three months) depending upon the application and its processing cycles. This usage information will be useful in determining the impact of an additional index or an additional column in an index. More explanation will be provided later on how to analyze changes to verify overall performance benefits.

DB2 has changed significantly between Version 7 and Version 8 the rules concerning when a predicate is indexable and when it is Stage 1. These changes can make a significant difference in performance of a DB2 application.

### Optimize Subqueries

A subquery is defined as the presence of a SELECT statement within the WHERE or HAVING clause of another SQL statement. Subqueries can be either correlated or non-correlated. A correlated subquery is one that refers to at least one column of the outer query. A non-correlated subquery does not refer to a column in the outer query. In some cases, DB2 will convert a subquery to a join. This would be indicated by the METHOD column in the PLAN\_TABLE table after execution of an EXPLAIN within DB2. There are times when DB2 does not make this conversion that it would still save resources to convert a subquery to a Join. Subqueries are as a general rule treated as a predicate by DB2 and would follow the same rules for determining whether the predicate is Stage 1 or 2. Subqueries result in multiple rows in the Explain Tables from DB2 that represent the multiple phases necessary to execute the query. As a general rule, these queries will perform better and use fewer resources when the column(s) referenced are indexable.

### Sequence of Table Access

When an SQL statement requires data from more than one table, the order of access can be critically important. The objective when accessing more than one table is to use the most restrictive table access first and continue until you access the least restrictive table last. This allows DB2 to eliminate from result set consideration as many rows as possible as early in the process as possible. This will result in reduced use of CPU and I/O resources.

All of the optimizer's object statistics are an important consideration when evaluating the sequence of table access. When these statistics are inaccurate, the likelihood of a poor sequence is significantly higher. Therefore, catalog statistics that match the actual data are important in guaranteeing correct sequence of table access.

The sequence chosen by the Optimizer can be verified for optimal performance through a simple process of query decomposition. For any query that accesses multiple tables, the user can extract the portion of the query for each table and convert the select to a count for the purposes of this exercise. The execution of each of the count statements will result in a single value for the number of qualifying rows for that portion of the query. You can sort the count queries in ascending order by the number of qualifying rows and compare that sequence to the Optimizer chosen sequence. When there are differences, then there are tuning opportunities.

## Changing the Schema

The second option for performance tuning is to change the schema.

The importance of indexing was mentioned earlier. Data concerning columns used in predicates provides the basis for determining the optimal index structure for an application.

Index design is affected by many factors:

- > Predicate usage
- > Read vs. update frequencies
- > Correlated columns
- > Impact on maintenance
- > Column values
- > DB2 limits

DB2 Version 8 provides more choices for index design. Non-partitioning secondary indexes (NPSI) have always existed in DB2, but with Version 8 you can now choose a data-partitioned secondary index (DPSI). For the primary index, you still choose to make it unique or not unique as well as making it clustered or not clustered. The new DPSI provides for secondary indexes through table-based partitioning.

DPSI provides for greater partition independence and as a result has great benefit for all utility executions. However, a DPSI is a resource-intensive choice in access path selection if the only predicates reference columns in the DPSI. This situation will result in an index scan of the index for each partition in the table unless there is partition screening present that has been mentioned earlier. Therefore, some

process should be put in place to examine the use of a DPSI to ensure that this one case does not exist.

## Predicate Usage

Index design must be based on predicate usage.

In the past, data analysts responsible for the logical design of an application would map out the entities (columns) in the application and estimate the number of times the data would be created, read, updated, and deleted. Because all those involved in IT are prone to acronyms, this process became known as the creation of a CRUD matrix. Designing the schema correctly from the beginning is always the best policy, but it is sometimes not an option. So, to deal with existing applications that may not have been designed optimally, it is sometimes beneficial to create this matrix after the fact. To create it with actual execution data is the best case possible. It was mentioned earlier that collection and analysis of predicate usage is important. The creation of a CRUD matrix for predicate usage provides this capability. This data provides a valid basis for the analysis of an existing index structure and the possible benefit of altered indexes as well as new indexes.

## Read vs. Update Frequencies

The predicate data can be aggregated to determine how often updates would occur for an existing index or a potentially new index. This information is important to know because index pages can become disorganized just like data pages – which leads to increased I/O activity, greater resource utilization, and decreased performance. This is one of the reasons why databases are reorganized. With the advent of DPSI and the improved concurrency during utility execution, even existing application schema should be examined to estimate the benefit, if any, of a DPSI.

## Correlated Columns

The predicate data can be used to find columns that are used together in SQL statements. If these columns are used often and accessed together, they become good candidates to participate in a multiple column index.

## Impact on Maintenance

Depending upon the application, there may be tables that are refreshed on a regular basis with a LOAD utility or there may be tables and/or indexes that are reorganized on a regular basis to improve performance. These activities must be taken into consideration when creating an index strategy for the application. As a result, there may be times when a tradeoff must be made between performance of the SQL versus the performance of the maintenance utilities.

## Column Values

Columns that have a limited number of values provide the Optimizer with significant opportunities for improved performance. When these columns appear in equality predicates, the Optimizer can make use of frequency statistics (more on these later) to accurately estimate filter factors and make a better index choice as a result. This is especially significant when the data for the column is skewed.

## DB2 Limits

Any index design must accommodate the limits imposed by DB2. With Version 8, the length of an index has been increased to 2000 bytes. This is a significant change from the 255 bytes in Version 7 and should provide for more options in index design/redesign in Version 8. In addition, DB2 prior to Version 8 would pad variable length columns in an index. Version 8 provides an option to store the variable length data in the index. These changes in DB2 should cause users to evaluate application index design in Version 8 to determine what changes can be made and of those possible changes which ones would provide benefit.

Many factors affect index design; however, the place to start is a matrix for columns referenced in SQL that is categorized by the type of access, frequency of access, and correlation of access. Consider the factors that have been mentioned earlier concerning tuning the SQL and the Optimizer's access path selection process. This data can lead to a more intelligent testing process that helps you arrive at a good index design.

## Can't Change the SQL

When a user cannot change the SQL text, index design/redesign is sometimes considered the only option for tuning SQL. That is not entirely true. In addition to index design, some performance can be gained by tuning the DB2 buffer pool configuration if it is not optimal. Finally, a third option is to collect statistics that will affect access path selection. More will be provided on this option later.

## Use a Hint

A third option for SQL performance tuning is to provide the Optimizer with a hint.

This may be looked upon as the brute force option. However, when a manager is looking over your shoulder to help you fix the current problem and users are calling for satisfaction, this option is very viable if only for a short period of time. Every effort should be made to keep the number of hints to as small a number as possible. Hints should be the subject of frequent analysis to find a way to replace the hint. Hints can be very effective, however, hints are difficult to create properly based on all the rules imposed by the Optimizer.

## Issues with Hints

The DB2 subsystem must be initialized to accept Optimizer hints. This can be accomplished by coding YES in the OPTIMIZATION HINTS field of the install panel DSNTIP4. Follow this change with a regeneration of the DSNZPARM member for the affected DB2 subsystem. Failure to set this parameter correctly will result in the Optimizer ignoring all hints whether they are valid or not.

Hints are tied to an SQL statement QUERYNO (the query number). One problem that can occur is that the query number in a package can change over time as new statements are added to the package. When the query number changes, the Optimizer will stop using the hint. The good news is a bind/rebind process will be the event that causes the Optimizer to stop using the hint. Therefore, an examination of bind error messages will be the alert for this situation.

Hints require the manual update of an existing row or rows in a PLAN\_TABLE table to create the hint. Because this process is manual, there is always a possibility for error.

Determining whether DB2 will use the hint depends upon whether the SQL is dynamic or static.

For dynamic SQL, you can choose between binding the package with the OPTHINT parameter or adding an SQL statement to set the CURRENT OPTIMIZATION HINT special register. SET CURRENT OPTIMIZATION HINT = hint name is an example of the SQL to accomplish this task. You can then Explain the statement to ensure that DB2 will use the hint. A return code of +394 indicates the hint will be used, and a return code of +395 indicates the hint is invalid. Additionally, if the dynamic statement cache is enabled, any occurrence of this statement in the cache would need to be invalidated. The simplest way to accomplish this is to execute a RUNSTATS utility against one of the objects referenced in the SQL statement. You do not have to collect statistics on Version 8 because that version supports the combination of REPORT (NO) and UPDATE (NO).

For static SQL, you must bind/rebind the package containing the SQL that requires the hint specifying the OPTHINT parameter with the hint name in parentheses. The same SQL return codes will apply for this bind. However, you should be careful to check the bind for changes to access paths for other statements in this package. Perhaps you will find opportunities for additional hints.

Another issue with this approach is the lack of debugging information when you encountered the dreaded +395 return code. This would be a good time to know someone who has been through this before so you can call and compare war stories.

## Changing or Collecting Object Statistics

The fourth and final option discussed in this paper for SQL performance tuning is the collection of statistics for the application objects.

Object statistics may be the most overlooked SQL tuning tool. The statistics collected and stored in the catalog provide the Optimizer with a better look at the reality of the objects being accessed and provide the Optimizer with a much better calculation of the cost for each possible access path. When DB2 chooses an unlikely access path that turns out to be resource intensive, the first place to look is at the object statistics to determine if they exist and if they are correct. The statistics stored in the catalog that influence access path selection can fall into three categories:

- > Cardinality
- > Distribution
- > Correlation

### Cardinality

Cardinality simply refers to a count of the number of rows in a given object. The referenced object could be a table, index, partition, or column. Cardinality is used quite often in cost calculations by the Optimizer.

Collection of the cardinality statistic is not costly, but because this statistic is a finite number, any change in the statistic could have an impact on the Optimizer cost calculations. Therefore, this statistic should be collected on a frequent periodic interval.

The Optimizer assumes a default cardinality of 10,000 for a table in the absence of this statistic. Therefore, this statistic should be calculated for all tables that have in excess of 10,000 rows.

The cardinality statistic is also influential in the selection of one index over another based on the number of qualifying rows and the number of pages in an index that will be read to retrieve the rows from the table. This is also a factor when the access path is index only and there are two indexes that can possibly provide the result set for the query.

### Distribution

Distribution statistics represent a count of the times a particular value appears in the object. The statistics collection utility of your choice allows for several options for distribution statistics. You can collect the most frequently occurring values, the least frequently occurring values, or both. The statistic is then stored in multiple rows in the catalog as a percentage value.

It is important to note that this is a percentage value. Unless the percentages change significantly because of a

merger/acquisition or some other company-changing event, this statistic can be collected once and remain valid until some significant data-changing event. Therefore, the cost of collecting this statistic is not nearly as important as the cost of cardinality statistics that are executed frequently.

Distribution statistics have a direct impact on the Optimizer when there is an equality or range predicate in the SQL. The Optimizer, by default, will assume an equal distribution of values for any particular column. If this default behavior does not reflect reality because the data is skewed to a small number of values, then the filter factor calculation of the Optimizer is inaccurate and the resulting access path may not be the most efficient one possible.

Perhaps an example would help make this clear. Let's assume a table named TABLE\_A exists. It has a column defined in the table named COLUMN\_A, which has a data type of decimal with possible values from 0 to 999. In this example, DB2 assumes an equal distribution of values and calculates a filter factor of .001 for any equal value(s) that might be referenced in the predicate of an SQL statement.

Let's assume that an SQL statement retrieves rows from this table with a range predicate in the format WHERE COLUMN\_A IS BETWEEN 0 AND 99. However, the range of values varies based on other input data from the user.

After the collection of distribution statistics, it is determined that the data is skewed as described in the following table:

Values for COLUMN_A	Frequency
0 to 99	5
100 to 199	0
200 to 299	20
300 to 399	0
400 to 499	25
500 to 599	0
600 to 699	40
700 to 799	0
800 to 899	5
900 to 999	5

For simplicity, the table does not contain every possible value.

When the Optimizer analyzes a predicate that requests the rows where COLUMN\_A is between 100 and 199, it will recognize immediately that no rows qualify and can plan for access path selection based on that information as opposed to the default assumption that 10 percent of the rows will qualify.

Furthermore, if this happens to be one of several predicates in the statement, then this filter factor when combined with all the other predicates will make a difference in the total

calculation of estimated rows. If the predicates are joined with an AND, the Optimizer will multiply the filter factors together and this one 0 for a filter factor would result in a filter factor of 0 for the statement as a whole. If the predicates are joined with an OR, the Optimizer would add the filter factors together and the impact of this 0 would not be as great.

## Correlation

Correlation statistics provide distribution statistics for a group of columns. The statistics are stored in the same table as the distribution statistics; the only difference in the values are distinct across a group of columns.

Perhaps an example would help. The United States is divided into zip codes for delivery of letters by the post office. There happens to exist a high degree of correlation between any particular city name and a zip code that might exist in that city. Therefore, any table that has a column for city as well as a column for zip code would be a candidate for correlation statistics.

Why is this important? When the Optimizer calculates the filter factor for the statement the formula will multiply any AND predicates together and will add any OR predicates together. If the SQL contains a WHERE CITY = 'some city name here' AND ZIP\_CODE = 'the same city's zip code here' the Optimizer will calculate a filter factor for the CITY column and multiply that by the filter factor for the zip code when, in reality, there is a high probability of a 1 to 1 correlation between the two values. This correlation can be determined by the collection of distribution statistics on a group of columns (like CITY and ZIP\_CODE). With these statistics present, DB2 will calculate a filter factor the statement correctly based on the actual data.

How are correlated columns detected? Many times these correlations are simply done during data analysis for the application. If that is not possible or was neglected, you can decompose any SQL statement that references two or more potentially related columns into a SELECT COUNT ..... SQL statement for each potentially related column in the statement. The closer the counts for the columns are to matching; then the closer is the correlation between the columns.

When columns are determined to be correlated, the correlation statistics can be collected for the related columns by using the COLGROUP parameter of the RUNSTATS utility.

## Optimize SQL Performance across the Application Lifecycle

BMC SQL Performance for DB2 helps IT reduce the cost of ownership for DB2 by tuning SQL statements across the application lifecycle. In development, BMC SQL Performance for DB2 helps developers and application DBAs evaluate SQL statements to ensure they meet best practices for SQL coding, which allows you to identify and resolve potential problems before they get to production.

In the test environment, BMC SQL Performance for DB2 helps application DBAs evaluate application performance metrics (how many resources each SQL statement actually takes). This helps resolve potential production problems earlier in the application lifecycle and reduces the cost of SQL tuning. Indexing strategies can also be evaluated to identify potential changes that could improve performance.

In production, BMC SQL Performance for DB2 identifies SQL statements that consume the most resources, and thus provides the greatest potential benefit from SQL tuning efforts. Once inefficient SQL statements are identified, the solution analyzes and offers recommendations on how to improve performance. You can model changes to SQL and DB2 objects to provide a cost/benefit analysis of proposed changes and see exactly how much you can save. Finally, BMC SQL Performance for DB2 can help you identify more effective indexing strategies based on production performance metrics.

## Key Features and Benefits

BMC SQL Performance for DB2 helps improve efficiency and reduce costs. The solution enables you to:

- > Quickly identify and tune SQL statements – Identifies the most expensive SQL statements and makes tuning recommendations that improve SQL performance, without costly SQL traces
- > Increase developer productivity and efficiency – Provides access path analysis and easy-to-use tuning tools for analyzing SQL statements as they are being developed, identifying problems earlier in the application lifecycle
- > Provide in-depth index optimization recommendations Includes "what-if " analysis and validation of alternative indexing strategies

BMC SQL Performance for DB2 offers a robust and easy-to-use solution to improve SQL performance, increase productivity, and control the total cost of ownership for DB2.

To learn more about BMC SQL Performance for DB2 products, attend one of our complimentary DB2 Technical Insights Webinars at [www.bmc.com/db2webevent](http://www.bmc.com/db2webevent).

## Conclusion

All of the changes mentioned must be verified through some “before and after” comparison process. The process is similar to what DB2 customers have been performing for the conversion from Version 7 to Version 8. Many customers that are currently running Version 7 and planning a migration to Version 8 have implemented a project to compare access path selection changes during the migration. Some of these changes are available in Version 8 compatibility mode, while some are not available until new function mode. Because of these changes within DB2, the project needs to include analysis in these two modes of Version 8. It is important to set up a valid comparison to ensure the accuracy and dependability of the results. This comparison is important because DB2 Version 8 has additional CPU overhead simply because of the changes to the DB2 programs. However, this overhead is offset by improvements in the processing of certain SQL statements. Therefore, this analysis is very important in the process of predicting the impact of upgrading to the new version of DB2. This does not imply that some will do this analysis to determine whether to upgrade to Version 8, but rather everyone should be in a position to set expectations for the performance of DB2 Version 8 so users will not be surprised.

A valid comparison can be hard to achieve. The DB2 Optimizer is affected by the subsystem, the CPU processing capacity, the buffer pool configurations in addition to the schema and actual SQL. How can all of these factors be managed to get a valid comparison? Some DB2 customers have used the ability of intelligent disk drives (Shark, Symmetrix or Lightning) to clone a production subsystem, upgrade to Version 8, and then execute DB2 Explains to generate Explain Tables that can be used for comparisons. There is a high price to accomplish such a realistic generation of Explain Tables that is only mitigated if the company has sufficient excess disk capacity to support such an implementation. Another approach is to use a test or QA subsystem which has been upgraded to DB2 Version 8 to conduct such a test. The production application schema is migrated to the test subsystem including in this process all production statistics. The buffer pool configuration is changed to match production. Then the explain process is executed to generate Explain Tables for comparison.

A third uncommon process is to use the DRBM library from the production subsystem as input to the execution of Explain processing on a development subsystem that has been upgrade to DB2 Version 8. This is not common, but it does have the added benefit of removing the need to migrate the application plan and packages structures from the production subsystem. The schema may still have to be migrated to the testing subsystem.

One important point to remember is that all of these processes assume that the SQL which is being Explained is exactly the same in both the Version 7 and the Version 8 subsystem. This is possible when the goal is to determine the difference between access path selections for each of the two versions of DB2. This is important to note because the comparison process is easier because there should be no differences in the SQL statement numbers on the two subsystems. In all companies where I have seen this attempted, the comparison has been accomplished with software as opposed to a manual comparison performed even by the smartest of DB2 professionals. The most difficult task in this process is the translation of access path changes to actual impact on resource utilization.

This process just described is also the process that can be used to determine the value of changes to an application’s SQL or schema. The same considerations apply to the comparison of access path selections and the ability to translate access path changes into the impact on resource utilization.

This paper will certainly not provide a person with everything necessary to tune DB2 at the SQL statement level, but it will provide a starting point for that analysis. This paper provides a basis for understanding the DB2 Optimizer and information that will assist you in making reasonable decisions regarding the source of the performance problem and the place to look in the data for clues to the root cause of the performance problem. With DB2 for z/OS, there are many opportunities for tuning SQL. Unfortunately, the starting point for each one may be different. Practice and experience are the best teachers for identifying where to start.

Any automation that can be built into your processes will be significantly beneficial, whether in collecting or analyzing real time data. Testing of potential solutions and benchmarking the improvements are good practices to document your work and to receive credit for your achievements.

## BMC SQL Performance for DB2

BMC® SQL Performance for DB2 helps you effectively address many of the SQL performance issues discussed in Tom Moulder’s whitepaper. This solution analyzes SQL statements and DB2 objects, identifies improvements that can dramatically improve SQL performance, and reduces the cost of ownership for DB2 applications.



ACTIVATE BUSINESS WITH THE POWER OF I.T.™

## About BMC Software

BMC Software, Inc. [NYSE:BMC], is a leading provider of enterprise management solutions that empower companies to manage their IT infrastructure from a business perspective. Delivering Business Service Management, BMC Software solutions span enterprise systems, applications, databases and service management. Founded in 1980, BMC Software has offices worldwide and fiscal 2004 revenues of more than \$1.4 billion. For more information about BMC Software, visit [www.bmc.com](http://www.bmc.com).

## About the Author

Tom Moulder is a recognized expert in the field of z/OS performance and tuning. His career includes over 30 years devoted to the tuning of z/OS and its related subsystems. He has published articles in magazines (most recently in the October 2005 issue of z/Journal), presented at conferences around the world (most recently at IDUG North America 2006 in Tampa, FL.) and served as an expert on the [www.search390.com](http://www.search390.com) web site. He is chairman of Southwest CMG, on the conference planning committee for national CMG this year and active in all aspects of CMG. Tom is a certified Database Administrator for DB2 for z/OS Version 8. Tom is a Certified Service Provider of BMC Software, Inc.

### TREX Associates, Inc.

Tom Moulder  
President  
9728 Delmonico Dr.  
Keller, TX. 76248  
+1 817 741-5549 Office  
+1 682 558-6527 Mobile  
[tom.moulder@trexassociates.com](mailto:tom.moulder@trexassociates.com)  
[www.t-rex-associates.com](http://www.t-rex-associates.com)

