Influence query optimization with optimization profiles and statistical views in DB2 9

Optimal query performance in DB2 9 for Linux, UNIX, and Windows

Kenneth K. Chen

December 07, 2006

Learn about enhancements in IBM® DB2® 9 for Linux®, UNIX®, and Windows® that enable you to influence the default query optimization behaviour. This capability is particularly useful when you are tuning classes of SQL queries that are not performing as expected, even though you have tuned your queries using the best practices. New techniques now make it possible to overwrite the optimizer behaviour and improve your query performance.

Introduction

Most major relational database management systems such as IBM DB2, Oracle and Microsoft® SQL Server rely on a cost-based optimizer design to choose an optimal SQL execution plan among the many possible alternative plans, under a set of often changing conditions within a database server environment, including changing query characteristics and changing data. In particular, DB2 SQL optimization decisions are influenced by the system configuration (I/O storage characteristics, CPU parallelism and speed, bufferpool and sortheap settings, communication bandwidth), schemas (indexes, constraints), DB2 registry variables, DB2 optimization level, and statistics information (about tables, columns, and indexes). The many complex factors and the dynamic nature of data means that an optimal plan evaluation is often a complex process indeed for any database system.

Considering the non-trivial task of generating an optimal SQL execution plan, DB2 continues to evolve its sophisticated cost model as well as adding new functionalities to provide the cost model with better information to make decisions. Statistical views are a powerful new form of statistics that represent relationships among complex predicates or across tables. The REOPT bind option defers query optimization until input variable values are available at OPEN time. The optimizer can then compare the input variable values to catalog statistics and compute a better selectivity estimate for predicates. Both statistical views and REOPT allow the optimizer to compute a more accurate cardinality estimate and consequently choose an optimal query execution plan. For the exceptional situations where the optimizer cannot choose an optimal query execution plan, DB2 has added features such as the SELECTIVITY clause and optimization guidelines.
In this discussion, we will look at the newest enhancements, the optimization guidelines and statistical views. You'll gain an understanding of what these enhancements do, and explore how they can be exploited in applications, in both non-data partitioning (non-DPF) and data partitioning (DPF) environments, under certain circumstances.

**DB2 optimization profiles**

In Version 8 FP9, DB2 for Linux, UNIX, and Windows incorporated optimization profile capability, where a guideline to generate a desired execution plan for an SQL query is passed to the optimizer, overriding the cost model that would have been used.

Many of us have encountered application situations where the majority of the query workload is properly tuned and is performing well, except for a handful of SQL statements, as a result of increasing user expectations and complex and diverse systems. Despite the best effort to tune the SQL statements through database changes such as use of index advisor or otherwise to improve indexing, updating statistics, improving data clustering, and making parameter changes, the problem may still persist. Sometimes we hope to influence the optimizer more directly, with minimum application changes if possible.

This is the time to explore the possibility that the optimization guidelines may be applicable. However, note that the state-of-the-art optimizer has a reason to generate a particular access plan, so it is important to ensure that we understand the inefficiency before we apply the guidelines. The use of guidelines is straightforward, but the more challenging task is to determine what is wrong with the SQL statements in the given database context and select the appropriate guidelines to apply.

**How optimization profiles work**

First choose a set of queries whose access plans you have decided to influence. These queries are then placed in an XML optimization profile along with the appropriate guidelines. The optimization profile must conform to the optimization guidelines XML schema for validation, and consists of a number of sections as shown in Listing 1.

**Listing 1. XML optimization profile**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.1.0.0">
  <!-- Optional zero or one Global section -->
  <OPTGUIDELINES>
    <MQT NAME="KCHEN.M1"/>
  </OPTGUIDELINES>
  <!-- Zero or more Statement profile section.-->  
  <STMTPROFILE ID="Query 0">
    <STMTKEY SCHEMA="KCHEN">
      <![CDATA[select c2 from t1 where c1=100]]>
    </STMTKEY>
    <!-- Guideline for Query 0-->
    <OPTGUIDELINES>
      <IXSCAN TABID="Q1" INDEX="T1X"/>
    </OPTGUIDELINES>
  </STMTPROFILE>
</OPTPROFILE>
```
The XML optimization profile starts with the OPTPROFILE section indicating the version attribute. The global section applies the rules globally to all SQL statements. You can specify for example what REOPT options to use, or what MQT table to consider or what query optimization to use. The statement profile section indicates what specific rules apply for individual SQL statements in the STMTKEY element.

The XML optimization profile is handy if the problem SQL query is not easily accessible. For example, it could be in an application that cannot be changed. In this case, we use the profile to affect the query behavior by triggering the guidelines associated with the query at runtime after successful query text matching. All SQL statements running in the environment will try to look for matches in the optimization profile that is active, and the matching is designed to be efficient with minimal overhead.

**How to enable an optimization profile**

It is possible to have many optimization profiles in a database, but in practice, it is more feasible to create one master optimization profile, with all the rules (statement profiles) grouped together, and activate the profile with one of the following methods depending on the choice of application environment. The DB2_OPTPROFILE registry variable also needs to be set to YES.

1. In the CLP environment: Use "SET CURRENT OPTIMIZATION PROFILE=KCHEN.PROF1" statement to associate the profile at the session level for any SQL statements, until connect reset, or profile reset. This statement can also be embedded in application programs.

2. For CLI application or JDBC applications using the legacy JDBC driver: Use the CURRENTOPTIMIZATIONPROFILE keyword setting in the db2cli.ini configuration file to associate the profile. Here for the SAMPLE database, the keyword is set under the data source section.

   ```
   [SAMPLE]
   CURRENTOPTIMIZATIONPROFILE=KCHEN.PROF1
   ```

   With this setting, the SQLs in the application execution will try to match with the SQL statements in KCHEN.PROF1 to look for specified rules which will override the general optimizations in the execution environment.

3. For JDBC application using the JCC Universal Driver: The JDBC application via the JCC Universal Driver does not use the DB2 CLI layer. While you can associate a system package and bind file with the dynamic SQL execution, it is best to embed the statement "SET CURRENT OPTIMIZATION PROFILE=KCHEN.PROF1" in the Java™ application to associate the profile at the session level.

4. For SQL PL procedures: Use SET_ROUTINE_OPTS procedure call before the creation of SQL PL procedures creation to associate the profile name with the specific SQL PL for DB2 V8 FP13+, or DB2 V9 FP1+.

   ```
   CALL SYSPROC.SET_ROUTINE_OPTS('OPTPROFILE KCHEN.PROF1')
   ```

   SQL PL procedures contain SQL statements that have execution attributes, such as the isolation level or the optimization level that can only be overridden by
DB2_SQLROUTINE_PREOPTS registry variable. You can also override the option with the SYSPROC.SET_ROUTINE_OPTS procedure. For a profile to be activated, this stored procedure can be used to associate the guidelines.

5. For embedded SQL in C/C++ applications:
   Use the OPTPROFILE bind option for embedded C/C++ applications. The embedded SQC program needs to be precompiled using the PREP command, which will create the bind file. This bind file needs to be bound with the OPTPROFILE option to the database, for example:
   
   ```
   bind progl.bnd OPTPROFILE KCHEN.PROF1
   ```

6. For SQLJ applications with embedded static SQL statements:
   Use the BINDOPTIONS parameter in the customize phase to associate the profile. The static SQLJ program, prog1, is translated and compiled as follows:
   
   ```
   sqlj progl.sqlj
db2sqljcustomize -url jdbc:db2://SERVER:PORT/SAMPLE -user USER -password PASSWORD -bindoptions "OPTPROFILE KCHEN.PROF1" -storebindoptions prog1_SJProfile0
   ```

All JDBC programs using legacy JDBC drivers will use the setting in db2cli.ini. The JDBC programs using the Universal JDBC driver will fall into category 3 above. It is important to note that since SQLJ generates an implicit "DECLARE CURSOR" clause for a SELECT SQL statement, it is necessary to also include the "DECLARE CURSOR" clause along with the SELECT statement in the optimization profile in order for the guideline to be picked up.

As the application executes, the SQL is compared with the guideline in the active profile. A matching STMTKEY, will allow the guidelines to kick in; otherwise, an SQL0437W with rc = 13 may be returned if, for example, the guideline is deemed inapplicable or invalid. The DB2 Explain facility is very useful to help in determining if the guideline is selected or not. The optimization profile name and the valid guideline will be indicated in the explain output. The guideline in the profile typically overrides the general optimization for the application settings, therefore giving good control of plan evaluation to the profile.

**Optimization guideline examples**

Any guidelines in the optimization profile must conform to the DB2 provided XML schema. If the guidelines are not correctly specified, then the guidelines will be ineffective, and in most cases, an SQL0437W with rc = 13 is returned. Optimization profiles are stored in a table called SYSTOOLS.OPT_PROFILE. If a guideline is updated, or deleted from the table, then it is necessary to refresh the cache by issuing the FLUSH OPTIMIZATION PROFILE CACHE statement, so it can be used. It is important to note that the SQL statement test matching is case sensitive but DB2 will try to remove redundant blanks and control characters before attempting to match.

The following examples are demonstrative of optimization profiles usage in 3 categories, namely, general optimization, query rewrite, and plan optimization.

**Example 1: Always use index T1X (plan optimization)**

Assume there is an index T1X on (c2, c1) columns of a table T1. For the following query, and depending on the optimizer costing, a tablescan can result. The following illustrates how to force an index.
Example 2: Use REOPT always (general optimization)

You can defer query optimization until input variables are known at runtime with the REOPT guideline. Possible options are ONCE, ALWAYS or NONE.

Example 3: Use Optimization Level 0 in DB2 V9 only (general optimization)

Typically, the optimization level is fixed for an application, but if there is a need to influence a particular SQL to execute at a different optimization level, you can create the following optimization guideline.

Example 4: Use Runtime degree ANY in DB2 V9 only (general optimization)

The runtime degree of a query for intra-partition can be altered in many ways. The following shows how an optimization guideline can specify the runtime degree for a query and affect the query execution.

Example 5: INLIST to Nested Loop Join (query rewrite)
Changing the list of values (inlist) to use the GENROW function can be very efficient and can improve the performance of the query. The list of values is put in a table in memory in this case.

Example 6: Sub-Query to Join (query rewrite)

Here a sub-query is transformed to a join during query rewrite using the SUBQ2JOIN with ENABLE attribute, so it can be better optimized.

Example 7: Influencing Join Order 3, 4, 1, 2 (plan optimization)

Often the join order of a query will greatly determine the query execution performance, since it is more efficient to filter rows as early as possible. The order can be influenced with the following guideline. Note the use of TABLEID attribute instead of TABID when multiple table references occur.
Influence query optimization with optimization profiles and statistical views in DB2 9

Example 8: A customer usage (plan optimization)

A customer experiences performance issue while refreshing an MQT as part of a batch run. The MQT refresh is triggered when the table tab2 involved in its definition is populated. The problem can be simulated in the following example code.

```
cREATE TABLE TAB1 (I INT, B VARCHAR(30))
CREATE TABLE TAB2 (I INT, B VARCHAR(150))
CREATE TABLE MQT1 (CNT, VAL) AS
    DATA INITIALLY DEFERRED REFRESH IMMEDIATE
CREATE INDEX I11 ON TAB1 (I ASC, B ASC)
CREATE INDEX I12 ON TAB1 (B ASC, I ASC)
CREATE INDEX I21 ON TAB2 (I ASC, B ASC)
CREATE INDEX I22 ON TAB2 (B ASC, I ASC)
INSERT INTO TAB2 VALUES(14, SUBSTR(CHAR(CURRENT TIMESTAMP), 1, 5))
```

In this scenario, after analysis, it was determined that it will be optimum to use the index I11 to access the table TAB1, but the default behaviour of the optimizer did not, even after tuning. The following guideline was created to influence the optimizer to consider I11 index, and substantially improved the time to refresh the MQT MQT1 by more than 2X.

```
<STMTPROFILE ID="Guideline for query 8">
    <STMTKEY>
        <![CDATA[INSERT INTO TAB2 VALUES(14, SUBSTR(CHAR(CURRENT_TIMESTAMP), 1, 5))]>
    </STMTKEY>
    <OPTGUIDELINES>
        <LPREFETCH TABLE="TAB1" INDEX="I11"/>
    </OPTGUIDELINES>
</STMTPROFILE>
```
Statistical views

Fundamentally, data in a relational database changes because of transactions and batch updates -- even a data mart, or data warehouse content will change over time. The SQL workloads are frequently dynamic SQL (as opposed to static), so any cost-based optimizer will generally have to make assumptions about data, data selectivity, and data cardinality, but very frequently the data distribution is unpredictably non-uniform, and nature of data domain values, and perhaps the inter-dependencies of tables and predicates make the optimizer liable to err.

Since the queries are dynamic, with unknown selection criteria at compile time, plan generation can be incorrect, even with the availability of full distribution statistics on the data. If the optimizer can peek into the query result (or partial query results), and armed with that foresight, then the information can be extremely useful to help determine a more accurate access plan.

Fundamentally there are two assumptions:

- Uniform distribution
- Domain values

To understand the case for statistical views, we first look at the above two assumptions, which may not be true in general. Hence, the need for statistical views arises in query plan optimization.

Uniform distribution

Consider the following data,

<table>
<thead>
<tr>
<th>C1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
</tbody>
</table>

where runstats (no distribution) would provide us the following information for C1:

CARD = 10,
COLCARD = 6,
LOW2KEY = 2,
HIGH2KEY = 9

Then:
- The number of rows with C1=3 will be estimated as 10/6 = 1.67
- The range of values between C1=4 and C1=8 is estimated to be ((8-4)/(9-2)) * 10 = 5.71

But if the data is changed to reflect non-uniformity and wide range of distribution:

<table>
<thead>
<tr>
<th>C1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>99</td>
</tr>
<tr>
<td>100</td>
</tr>
</tbody>
</table>

Then:
- The number of rows with C1=3 will be estimated as 10/6 = 1.67
- The range of values between C1=4 and C1=8 is estimated to be ((8-4)/(99-2)) * 10 = 0.41

And if the data is completely uniform as follows:

<table>
<thead>
<tr>
<th>C1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
</tbody>
</table>

Then:
- The number of rows with C1=3 will be estimated as 10/10 = 1
- The range of values between C1=4 and C1=8 is estimated to be ((8-4)/(9-2)) * 10 = 5.71
- The range of values between C1=3 and C1=7 is estimated to be ((7-3)/(9-2)) * 10 = 5.71

So when the data is uniform, the actual result and the estimated result is more consistent irrespective of the value and the ranges.
Even if we have distribution statistics like frequent values and quantile values, which greatly reduce the estimation errors for equality and range predicates, there are possible situations where the errors in estimation can still be unacceptable.

**Domain values**

a) Now if we extend to a two table join T1.C1 = T2.C1 for the following data, where one set of data encompasses the other set:

<table>
<thead>
<tr>
<th>T1.C1</th>
<th>T2.C1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

The selectivity of a predicate is defined as:
Selectivity = 1 / ( max ( C1 colcard , C2 colcard ) ) = 0.1

The cardinality is 10 * 10 * 0.1 = 10.

b) If the data in the two table join T1.C1 = T2.C1 differs slightly where the sets of data intersect, and one does not contain the other:

<table>
<thead>
<tr>
<th>T1.C1</th>
<th>T2.C1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>8</td>
<td>13</td>
</tr>
<tr>
<td>9</td>
<td>14</td>
</tr>
<tr>
<td>10</td>
<td>15</td>
</tr>
</tbody>
</table>
Here T1.C1 value, such as 7 will never join with T2.C1, or T2.C1 value, such as 12, will not join with T1.C1, but the estimation algorithm will not be aware of this, and will make inaccurate assumption that a value in T1 is just as likely to join with any value in T2 or vice versa.

The cardinality is still $10 \times 10 \times 0.1 = 10$.

So the costs are the same, but the actual rows output result for a) is 10, and for b) is 6.

<table>
<thead>
<tr>
<th>RESULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

Obviously, there are inconsistencies here and the estimation problem could potentially worsen with more complex joins. DB2 statistical view feature available from V8 FP9 onwards is designed to provide compensation for this class of inaccuracies due to data distribution and values.

To understand how statistical view can help, consider a slightly more realistic join scenario,

<table>
<thead>
<tr>
<th>T1.C1</th>
<th>T1.C2</th>
<th>T2.C1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>D</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>E</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>F</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>G</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>H</td>
<td>13</td>
</tr>
<tr>
<td>9</td>
<td>I</td>
<td>14</td>
</tr>
<tr>
<td>10</td>
<td>J</td>
<td>15</td>
</tr>
</tbody>
</table>

In this case, the join predicates of T1.C1=T2.C1 and T1.C2='A' (or any values of C2) will return a cardinality estimate of 1. But this is significantly wrong if the local predicate were T1.C2='B' or if T1.C2='E' as shown in the result below from the join of 2 above tables on T1.C1 = T2.C1
To compensate for the estimation error, a statistical view can be created and readied for exploitation as follow:

```sql
Create view SCHEMA.V1 as select * from T1, T2 where T1.C1 = T2.C1
Alter view SCHEMA.V1 enable query optimization
Runstats on table SCHEMA.V1 with distribution
```

For statistical view, the ENABLE QUERY OPTIMIZATION clause will cause the view, and its associated statistics to be used to improve the query optimization. It is only necessary to collect data runstats information, with distribution characteristics. The runstats information is the key to the statistical view deployment, and must provide more complete information than the base tables. Sometimes, column group, or like statistics options can be useful.

Now, the statistical view will contain the statistics information about data distribution after the join collected on the entire result set, which is complete in both the non-DPF as well as DPF environment, without any extrapolation. The runstats may sometimes take more time, which could possibly be due to a poor plan of the view itself. After the runstats, the following additional information is known:

- The COLCARD of column of the result
- The cardinality of the result
- Values, and count of values

This information can then be included as part of the optimizer decision during costing for selectivity and cardinality estimates for qualified queries which do not need to reference the view directly. This will result in more accurate costing, and more optimal access plan.

Using the same sample data just discussed above, the following illustrates the difference in cardinality estimates with and without the use of statistical view for the query.

```sql
select * from T1,T2 where T1.C2='B' and T1.C1=T2.C1
```

1) No statistical view - Here the cardinality based on the data is off significantly. The hash join in the access plan estimates that 1 row will be returned when in fact it is 4 rows.
Influence query optimization with optimization profiles and statistical views in DB2 9

Listing 2. Access Plan with no statistical view

<table>
<thead>
<tr>
<th>Rows RETURN</th>
<th>Cost</th>
<th>I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 1)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>HSJOIN</td>
<td>15.1653</td>
<td></td>
</tr>
<tr>
<td>( 2)</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

/-----+-----\
10     1
TBSCAN TBSCAN
( 3)   ( 4)
7.58162 7.58301
1       1
|       |
10     10
TABLE: TABLE:
KCHEN.T2 KCHEN.T1

2) Statistical view - Here the cardinality based on the data is 4 rows, which is highly improved over case 1, and accurate in this case.
Note that the explain output will contain the following diagnostic information to indicate that the statistical view is in use. In this regard, db2exfmt tool is very handy to determine whether a statistical view is used or not.
Diagnostic Details: EXP0147W. The following statistical statistical view may have been used by the optimizer to estimate cardinalities: "KCHEN "."V1".

Listing 3. Access plan with statistical view

<table>
<thead>
<tr>
<th>Rows RETURN</th>
<th>Cost</th>
<th>I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 1)</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>HSJOIN</td>
<td>15.1653</td>
<td></td>
</tr>
<tr>
<td>( 2)</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

/-----+-----\
10     1
TBSCAN TBSCAN
( 3)   ( 4)
7.58162 7.58301
1       1
|       |
10     10
TABLE: TABLE:
KCHEN.T2 KCHEN.T1

The exploitation of statistical view containing the additional information of the join (with the runstats by executing the query during runstats, but does not materialize the actual detailed result set persistently) improved the cardinality estimates. The use of statistical view for one or more queries is transparent, without having to directly reference to it. The statistical view is kind of like the
Materialized Query Table (MQT), except that there is no need for materialization. In fact, the support of statistical view shares some similar limitations to the support of MQT in DB2. The statistical view currently does not support aggregate functions like SUM, MAX, distinct operations or set operation like UNION, EXCEPT or INTERSECT.

In DB2 V8 FP9, the registry variable setting DB2_STATVIEW is needed, and there is a constraint that only 2 table references can be in the statistical view. In addition, manual collection of statistics is needed, since runstats do not work on the statistical view. In DB2 V9, all of these limitations have been lifted.

Generally, in a star join scenario involving fact table and many dimension tables, the columns of the dimension tables are included (the local predicate columns are particularly important), the fact table columns are not necessary. This is because the data distribution for fact table columns does not change when joined with the dimension tables so the optimizer can get an accurate selectivity estimate with just the distribution statistics on the fact table columns. An exception to this is in V8 where fact table columns referenced in any query predicate must be included in the statistical view definition due to MQT routing limitations.

Given a 3 table join, it is possible to create multiple statistical views i.e. on T1 and T2, T1, and T3, and T1, T2, T3 which will have all the statistics information to generate the best access plan, even though only the first 2 are sufficient.

Example 1:

Select * from T1, T2, T3 where T1.C1=T2.C1 and T1.C2=T3.C2 and T2.C1=2 and T3.C2 = 'B'

Create view SCHEMA.V11 as select T2.* from T1, T2 where T1.C1 = T2.C1

Create view SCHEMA.V12 as select T3.* from T1, T3 where T1.C2 = T3.C2

In the above, T1 is the fact table with FK-PK relationship with dimension tables T2 and T3. The two statistical views, V11 and V12 are created to improve estimation.

**Conclusion**

Optimization guidelines and statistical views are useful features to compensate the access plan evaluations when exceptions in query performance occur. These features can be considered after application of all standard tuning techniques but have not produced the desired result. They should be considered carefully, as there is an associated cost of query matching for guidelines, or compilation for statistical views. In addition, the specifics of the guidelines and statistical views may vary over time, as a result of the change of the state of the database, for example, the guidelines may change because the volume of data has changed, or the statistical view statistics are outdated. It is important to review their implementation regularly, so that the maximum benefit can be derived from their usage.
Acknowledgement

I would like to acknowledge John Hornibrook, DB2 query optimizer development manager, whose valuable input greatly help to improve the content accuracy and Daniel Deaconu, DB2 developer, who helped proofread the article and offered valuable suggestions.
Related topics

- Download a free trial version of **DB2 Enterprise 9**.
- Now you can use DB2 for free. Download **DB2 Express-C**, a no-charge version of DB2 Express Edition for the community that offers the same core data features as DB2 Express Edition and provides a solid base to build and deploy applications.
- Visit the developerWorks resource page for DB2 for Linux, UNIX, and Windows to read articles and tutorials and connect to other resources to expand your DB2 skills.
- Download **IBM product evaluation versions** and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

© Copyright IBM Corporation 2006
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)