TECHNICAL WHITEPAPER

bmcsoftware
ACTIVATE BUSINESS WITH THE POWER OF I.T.™

Exploiting Explain

Using Explain Information for Optimal Query Performance

Susan Lawson and Dan Luksetich
www.db2expert.com

YL&A®
www.ylassoc.com

About the Authors

Susan Lawson and Dan Luksetich of YL&A have a combined DB2 experience of over 38 years. They work with several large DB2 customers developing, deploying and tuning some of the world's largest and most complex databases.

Table of Contents

Introduction

Performing Explain on SQL statements is a critical part of application performance tuning. Without Explain information we would have a really hard time diagnosing SQL performance problems or determining if changes to SQL statements would result in a better access path. Prior to Version 8 we only had the PLAN_TABLE, DSN_FUNCTION_TABLE and the DSN_STATEMNT_TABLE. In V8/9 we got 11 new Explain tables and the DSN_STATEMENT_CACHE table.

In this paper we will describe both the old and the new Explain tables and demonstrate how to use the information in them to diagnose problems and to proactively tune SQL. We will also provide some queries showing how to use the new Explain tables in conjunction with the PLAN_TABLE to produce details regarding the access path and other valuable information. We will then take a look at what it takes to properly tune SQL and to accurately evaluate impacts of changes due to release migrations, statistics change or structure changes.

This paper provides you with a resource for all Explain tables, their usage and helpful tips and queries for using the information contained in them.

Explain

Explain is not an optimization enablement feature, but more of a tool to provide us with information about the potential optimization of our query.

Performing Explains

We must first perform the Explain by using one of the following methods:

- Binding our package with EXPLAIN(YES)
- Performing an Explain on an individual statement with EXPLAIN using the following syntax:

```
EXPLAIN PLAN FOR sql-statement SET QUERYNO=integer
```

- Performing an Explain on a statement in the dynamic SQL cache using the following syntax:

```
EXPLAIN STMTCACHE STMTID=int (or STMTTOKEN=string)
```

That is the easy part. The hard part is yet to come when you interpret the output in the Explain tables to determine if the access path is optimal.

Just Explain information is not truly enough to say if the query is optimal. We also need to have the cardinalities of our tables and indexes, we also need to understand what the query is trying to accomplish and we also need time. Tools are often helpful in this regard, since they can pull this information together and present it to us as well as give some recommendations. In this paper we will look at how to pull the most useful information from the Explain tables and how to interpret the results.

Factors Influencing Explain Output

We must also realize that Explain data is not a guarantee of what will actually happen at run time. Other factors can also influence the final access path. The number of rows, the number of distinct values in the columns of the index, cluster ratio, the type of indexes, index uniqueness, the number of levels of the index tree, buffer pool size, sort pool size, RID pool size, CPU speed, and whether RUNSTATS has been run are some of the factors. Because of these factors, you need for the testing subsystem to look as much like the production system as possible. In order to accomplish this you have two alternatives. One is to define your test system exactly as the production system. This includes having production size data volume on test as well as having the DSNZPARM values set the same. A better alternative, would be to load a representative sampling of production data into the tables, have the buffer pools and other memory areas defined the same, have identical indexes, and update the DB2 Catalog with statistics from production.

The Explain Tables

We will take a look at the tables that are populated by Explain. These tables must first be created either by you executing the DDL or by invoking the Visual Explain or Optimization Service Center products. Once these tables are created, they will be populated whenever an Explain is executed either by a tool, a bind, the Explain statement, or the DSN8EXP stored procedure (which can allow you to Explain against SQL you may not have authority to execute).

PLAN_TABLE

The PLAN_TABLE is the main table for Explain. It is populated with information about the access path chosen by the optimizer. The following shows all the columns in the PLAN_TABLE as well as a description of each column:

Table 1 PLAN_TABLE

Column Name	Description
QUERYNO	A number intended to identify the statement being explained. For a row produced by an EXPLAIN statement, specify the number in the QUERYNO clause. For a row produced by non-EXPLAIN statements, specify the number using the QUERYNO clause, which is an optional part of the SELECT, INSERT, UPDATE and DELETE statement syntax. Otherwise, DB2 assigns a number based on the line number of the SQL statement in the source program. FETCH statements do not each have an individual QUERYNO assigned to them. Instead, DB2 uses the QUERYNO of the DECLARE CURSOR statement for all corresponding FETCH statements for that cursor. When the values of QUERYNO are based on the statement number in the source program, values greater than 32767 are reported as 0. Hence, in a very long program, the value is not guaranteed to be unique. If QUERYNO is not unique, the value of TIMESTAMP is unique.
QBLOCKNO	A number that identifies each query block within a query. The value of the numbers are not in any particular order, nor are they necessarily consecutive.
APPLNAME	The name of the application plan for the row. Applies only to embedded EXPLAIN statements executed from a plan or to statements explained when binding a plan. Blank if not applicable.
PROGNAME	The name of the program or package containing the statement being explained. Applies only to embedded EXPLAIN statements and to statements explained as the result of binding a plan or package. Blank if not applicable.
PLANNO	The number of the step in which the query indicated in QBLOCKNO was processed. This column indicates the order in which the steps were executed.
METHOD	A number (0, 1, 2, 3, or 4) that indicates the join method used for the step: 0 First table accessed, continuation of previous table accessed, or not used. 1 Nested loop join. For each row of the present composite table, matching rows of a new table are found and joined. 2 Merge scan join. The present composite table and the new table are scanned in the order of the join columns, and matching rows are joined. 3 Sorts needed by ORDER BY, GROUP BY, SELECT DISTINCT, UNION, a quantified predicate, or an IN predicate. This step does not access a new table. 4 Hybrid join. The current composite table is scanned in the order of the join-column rows of the new table. The new table is accessed using list prefetch.
CREATOR	The creator of the new table accessed in this step, blank if METHOD is 3.
TNAME	The name of a table, materialized query table, created or declared temporary table, materialized view, or materialized table expression. The value is plan if METHOD is 3. The column can also contain the name of a table in the form DSNWFQB(qblockno). DSNWFQB(qblockno) is used to represent the intermediate result of a UNION ALL or an outer join that is materialized. If a view is merged, the name of the view does not appear.
TABNO	Values are for IBM use only.

Table 1 PLAN_TABLE

Column Name	Description
ACCESSTYPE	The method of accessing the new table: I By an index (identified in ACCESSCREATOR and ACCESSNAME) I1 By a one-fetch index scan M By a multiple index scan (followed by MX, MI, or MU) MX By an index scan on the index named in ACCESSNAME MI By an intersection of multiple indexes MU By a union of multiple indexes N By an index scan when the matching predicate contains the IN keyword R By a table space scan RW By a work file scan of the result of a materialized user defined table function T By a spare index (star join work files) V By buffers for an INSERT statement within a SELECT blank Not applicable to the current row
MATCHCOLS	For ACCESSTYPE I, I1, N or MX, the number of index keys used in an index scan; otherwise, 0.
ACCESSCREATOR	For ACCESSTYPE I, I1, N, or MX, the creator of the index; otherwise, blank.
ACCESSNAME	For ACCESSTYPE I, I1, N, or MX, the name of the index; otherwise, blank.
INDEXONLY	Whether access to an index alone is enough to carry out the step, or whether data too must be accessed. Y=Yes; N=No
SORTN_UNIQ	Whether the new table is sorted to remove duplicate rows. Y=Yes; N=No.
SORTN_JOIN	Whether the new table is sorted for join method 2 or 4. Y=Yes; N=No.
SORTN_ORDERBY	Whether the new table is sorted for ORDER BY. Y=Yes; N=No.
SORTN_GROUPBY	Whether the new table is sorted for GROUP BY. Y=Yes; N=No.
SORTC_UNIQ	Whether the composite table is sorted to remove duplicate rows. Y=Yes; N=No.
SORTC_JOIN	Whether the composite table is sorted for join method 1, 2 or 4. Y=Yes; N=No.
SORTC_ORDERBY	Whether the composite table is sorted for an ORDER BY clause or a quantified predicate. Y=Yes; N=No.
SORTC_GROUPBY	Whether the composite table is sorted for a GROUP BY clause. Y=Yes; N=No.
TSLOCKMODE	An indication of the mode of lock to be acquired on either the new table, or its table space or table space partitions. If the isolation can be determined at bind time, the values are: IS Intent share lock IX Intent exclusive lock S Share lock U Update lock X Exclusive lock SIX Share with intent exclusive lock N UR isolation; no lock If the isolation cannot be determined at bind time, then the lock mode determined by the isolation at run time is shown by the following values: NS For UR isolation, no lock; for CS, RS, or RR, an S lock. NIS For UR isolation, no lock; for CS, RS, or RR, an IS lock. NSS For UR isolation, no lock; for CS or RS, an IS lock; for RR, an S lock. SS For UR, CS, or RS isolation, an IS lock; for RR, an S lock. The data in this column is right-justified. For example, IX appears as a blank followed by I followed by X. If the column contains a blank, then no lock is acquired.
TIMESTAMP	Usually, the time at which the row is processed, to the last .01 second. If necessary, DB2 adds .01 second to the value to ensure that rows for two successive queries have different values.
REMARKS	A field into which you can insert any character string of 762 or fewer characters.
PREFETCH	Whether data pages are to be read in advance by prefetch. S = pure sequential prefetch; L = prefetch through a page list; D = optimizer expects dynamic prefetch; blank = unknown at bind time or no prefetch.
COLUMN_FN_EVAL	When an SQL column function is evaluated. R = while the data is being read from the table or index; S = while performing a sort to satisfy a GROUP BY clause; blank = after data retrieval and after any sorts.
MIXOPSEQ	The sequence number of a step in a multiple index operation. 1, 2, n. For the steps of the multiple index procedure (ACCESSTYPE is MX, MI, or MU.) 0. For any other rows (ACCESSTYPE is I, I1, M, N, R, or blank).
VERSION	The version identifier for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. Blank if not applicable.

Table 1 PLAN_TABLE

Column Name	Description
COLLID	The collection ID for the package. Applies only to an embedded EXPLAIN statement executed from a package or to a statement that is explained when binding a package. Blank if not applicable. The value DSNDYNAMICSQLCACHE indicates that the row is for a cached statement.
ACCESS_DEGREE	The number of parallel tasks or operations activated by a query. This value is determined at bind time; the actual number of parallel operations used at execution time could be different. This column contains 0 if there is a host variable.
ACCESS_PGROUP_ID	The identifier of the parallel group for accessing the new table. A parallel group is a set of consecutive operations, executed in parallel, that have the same number of parallel tasks. This value is determined at bind time; it could change at execution time.
JOIN_DEGREE	The number of parallel operations or tasks used in joining the composite table with the new table. This value is determined at bind time and can be 0 if there is a host variable. The actual number of parallel operations or tasks used at execution time could be different.
JOIN_PGROUP_ID	The identifier of the parallel group for joining the composite table with the new table. This value is determined at bind time; it could change at execution time.
SORTC_PGROUP_ID	The parallel group identifier for the parallel sort of the composite table.
SORTN_PGROUP_ID	The parallel group identifier for the parallel sort of the new table.
PARALLELISM_MODE	The kind of parallelism, if any, that is used at bind time: I Query I/O parallelism C Query CP parallelism X Sysplex query parallelism
MERGE_JOIN_COLS	The number of columns that are joined during a merge scan join (Method = 2).
CORRELATION_NAME	The correlation name of a table or view that is specified in the statement. If there is no correlation name, then the column is blank.
PAGE_RANGE	Whether the table qualifies for page range screening, so that plans scan only the partitions that are needed. Y=Yes; blank=No.
JOIN_TYPE	The type of an outer join. F FULL OUTER JOIN L LEFT OUTER JOIN S STAR JOIN blank INNER JOIN or no join RIGHT OUTER JOIN converts to a LEFT OUTER JOIN when you use it, so that JOIN_TYPE contains L.
GROUP_MEMBER	The member name of the DB2 that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data-sharing environment when EXPLAIN was executed.
IBM_SERVICE_DATA	Values are for IBM use only.
WHEN_OPTIMIZE	When the access path was determined: Blank – At bind time, using a default filter factor for any host variables, parameter markers, or special registers. B – At bind time, using a default filter factor for any host variables, parameter markers, or special registers; however, the statement is reoptimized at run time using input variable values for input host variables, parameter markers, or special registers. The bind option REOPT(ALWAYS) or REOPT(ONCE) must be specified for reoptimization to occur. R – At run time, using input variables for any host variables, parameter markers, or special registers. The bind option REOPT(ALWAYS) or REOPT(ONCE) must be specified for this to occur.
QBLOCK_TYPE	For each query block, the type of SQL operation performed. For the outermost query, it identifies the statement type. Possible values: SELECT SELECT INSERT INSERT UPDATE UPDATE DELETE DELETE SELUPD SELECT with FOR UPDATE OF DELCUR DELETE WHERE CURRENT OF CURSOR UPDCUR UPDATE WHERE CURRENT OF CURSOR CORSUB Correlated subquery NCOSUB Non-correlated subquery TABLEX Table Expression TRIGGR WHEN clause on CREATE TRIGGER UNION UNION UNIONA UNION ALL

Table 1 PLAN_TABLE

Column Name	Description
BIND_TIME	The time at which the plan or package for this statement or query block was bound. For static SQL statements, this is a full-precision timestamp value. For dynamic SQL statements, this is the value contained in the TIMESTAMP column of PLAN_TABLE appended by 4 zeroes.
OPTHINT	A string that you use to identify this row as an optimization hint for DB2. DB2 uses this row as input when choosing an access path.
HINT_USED	If DB2 used one of your optimization hints, it puts the identifier for that hint (the value in OPTHINT) in this column.
PRIMARY_ACCESSTYPE	Indicates whether direct row access will be attempted first: D DB2 will try to use direct row access. If DB2 cannot use direct row access at runtime, it uses the access path described in the ACCESSTYPE column of PLAN_TABLE. blank DB2 will not try to use direct row access.
PARENT_QBLOCK	Number that indicates the QBLOCKNO of the parent query
TABLE_TYPE	The type of new table: B = Buffers for an INSERT statement within a SELECT C = Common Table Expression F = Table Function M = Materialized Query Table Q = Temporary Intermediate Result table (not materialized). R = Recursive common table expression T = Table W = Work file
TABLE_ENCODE	The encoding scheme of the table. If the table has a single CCSID set, possible values are: A = ASCII E = EBCDIC U = Unicode M is the value of the column when the table contains multiple CCSID sets.
TABLE_SCCSID	SBCS CCSID value of the table. If column TABLE_ENCODE is M, value is 0.
TABLE_MCCSID	Mixed CCSID value of the table. If column TABLE_ENCODE is M, value is 0.
TABLE_DCCSID	DBCS CCSID value of the table. If column TABLE_ENCODE is M, value is 0.
ROUTINE_ID	Values for IBM use only.
CTREF	If the referenced table is a common table expression, the value is the top-level query block number.
STMTTOKEN	User-specified statement token.

As you can see, the PLAN_TABLE is rather large and you will need a query to format it so it can be more readable. The following query will provide an easy way to format the output (this query can be downloaded at WWW.DB2EXPERT.COM):

PLAN_TABLE Query

This query selects and formats the output from Explain that is in the PLAN_TABLE. It contains information about what tables are accessed, the type of access, indexability, sorting, prefetch type, parallelism, join type and hint information. It is then sorted by the program name, the bind time, query block number/plan number, and the multi-index sequence (if there is multi-index access).

```
SELECT SUBSTR(DIGITS(QUERYNO),5) CONCAT '-' CONCAT SUBSTR(DIGITS(QBLOCKNO),4) CONCAT '-'
CONCAT SUBSTR(DIGITS(PLANNO),4) AS Q_QB_PL ,PROGNAME AS PNAME ,SUBSTR(CHAR(METHOD),1,1) AS
MT ,SUBSTR(TNAME,1,18) AS TNAME ,CHAR(TABNO) AS T_NO ,ACCESSTYPE AS AT ,CHAR(MATCHCOLS) AS
MC ,SUBSTR(ACCESSNAME,1,8) AS ACC_NM ,INDEXONLY AS IX ,SORTN_JOIN CONCAT SORTC_UNIQ CONCAT
SORTC_JOIN CONCAT SORTC_ORDERBY CONCAT SORTC_GROUPBY AS NJ_CUJOG ,PREFETCH AS PF
,COLUMN_FN_EVAL AS CFE ,CHAR(MIXOPSEQ) AS MIX ,ACCESS_DEGREE AS A_DEG ,JOIN_DEGREE AS
J_DEG ,PARALLELISM_MODE AS P_MODE ,MERGE_JOIN_COLS AS MJC ,CORRELATION_NAME AS COR_NM
,PAGE_RANGE AS PG_RG ,JOIN_TYPE AS JT ,WHEN_OPTIMIZE AS WH_OP ,QBLOCK_TYPE AS QB_TYP
,BIND_TIME AS B_TM ,HINT_USED AS HINT ,PRIMARY_ACCESSTYPE AS PR_ACC
FROM PLAN_TABLE A
--WHERE PROGNAME = 'YLAPROG1' <-- TARGET A SPECIFIC PROGRAM HERE
WHERE BIND_TIME = (SELECT MAX(BIND_TIME) FROM PLAN_TABLE B WHERE A.PROGNAME = B.PROGNAME
AND A.COLLID = B.COLLID)
ORDER BY PROGNAME, BIND_TIME, Q_QB_PL, MIX;
```

The following shows a sample output from this query:

```
PROGNAME  QQP         MTH MJC TBCREATOR TBNAME               CORR_NM  ATYP A_NM            IXO MIX    MCOL
--------  ----------- --- --- --------- -------------------  -------- ---- --------------- --- ------ ------
DSNESM68  00003-01-01 0   --- DSN8710   EMP                  A        I    XEMP1           N   0      1
DSNESM68  00003-01-02 1   --- DSN8710   DEPT                 B        I    XDEPT1          N   0      1

NJ_CUJOG PF CFE PGRNG JT QB_TYP P_QB TB_TYP B_TM
-------- -- --- ----- -- ------ ---- ------ --------------------------
NNNNN                   SELECT    0 T      2007-08-13 15:47:24.150000
NNNNN                   SELECT    0 T      2007-08-13 15:47:24.150000
```

You can see in this output that there are two tables (EMP and DEPT) that are being joined together with a nested loop join, using two indexes (XEMP1 and XDEPT1) both matching on 1 column and there is no sorting.

STATEMENT and FUNCTION Tables

The DSN_STATEMNT_TABLE and the DSN_FUNCTION_TABLE were introduced sometime ago in V6. The DSN_STATEMNT_TABLE provides us with information about the estimated cost of executing the SQL statement.

DSN_STATEMNT_TABLE

The following shows the columns in the DSN_STATEMNT_TABLE. The most useful columns in the DSN_STATEMNT_TABLE are the COST_CATEGORY, PROCMS, PROCMU and REASON. You can get an idea from this information of the relative cost estimate for the query and whether DB2 had to use default values during the estimation and why (REASON). While this information can be useful, it is not a guarantee that just because a one query has a lower statement cost that another that it is necessarily better. In other words, this information should not be used alone to determine query performance.

Table 2 DSN_STATEMNT_TABLE

Column Name	Description
QUERYNO	A number intended to identify the statement being explained. If QUERYNO is not unique, the value of EXPLAIN_TIME is unique.
APPLNAME	The name of the application plan for the row, or blank.
PROGNAME	The name of the program or package containing the statement being explained, or blank.
COLLID	The collection ID for the package. Applies only to an embedded EXPLAN statement executed from a package or to a statement that is being explained when binding a package. Blank is not applicable. The value DSNDYNAMICSQLCACHE indicates that the row is for a cached statement.
GROUP_MEMBER	The member name of the DB2 that executed EXPLAIN, or blank.
EXPLAIN_TIME	The time at which the statement is processed. This time is the same as the BIND_TIME column in PLAN_TABLE.
STMT_TYPE	The type of statement being explained. Possible values: SELECT = SELECT INSERT = INSERT UPDATE = UPDATE DELETE = DELETE SELUPD = SELECT with FOR UPDATE OF DELCUR = DELETE WHERE CURRENT OF CURSOR UPDCUR = UPDATE WHERE CURRENT OF CURSOR
COST_CATEGORY	Indicates if DB2 was forced to use default values when making its estimates. Possible values: A = Indicates that DB2 had enough information to make a cost estimate without using default values. B = Indicates that some condition exists for which DB2 was forced to use default values. See the values in REASON to determine why DB2 was unable to put this estimate in cost category A.

Table 2 DSN_STATEMNT_TABLE

Column Name	Description
PROCMS	The estimated processor cost in milliseconds for the SQL statement. The estimate is rounded up to the next integer value. The maximum value for this cost is 2,147,483,647 milliseconds, which is equivalent to approximately 24.8 days. If the estimated value exceeds this maximum, the maximum value is reported. STMT_ENCODEThe encoding scheme of the statement. If the represents a single CCSID set, possible values are: A = ASCII E = EBCDIC U = Unicode If the statement has multiple CCSID sets, the value is M.
PROCSU	The estimated processor cost in service units for the SQL statement. The estimate is rounded up to the next integer value. The maximum value for this cost is 2,147,483,647 service units. If the estimated value exceeds this maximum, the maximum value is reported.
REASON	A string that indicates the reasons for putting an estimate into cost category B. HAVING CLAUSE: A subselect in the SQL statement contains a HAVING clause. HOST VARIABLES: The statement uses host variables, parameter markers, or special registers. REFERENTIAL CONSTRAINTS: Referential constraints of the type CASCADE or SET NULL exist on the target table of a DELETE statement. TABLE CARDINALITY: The cardinality statistics are missing for one or more of the tables used in the statement. Or, the statement required the materialization of views or nested table expression. UDF: The statement uses user-defined functions. TRIGGERS: Triggers are defined on the target table of an INSERT, UPDATE, or DELETE statement.
STMT_ENCODE	The encoding scheme of the statement. If the represents a single CCSID set, possible values are: A = ASCII E = EBCDIC U = Unicode If the statement has multiple CCSID sets, the value is M.

DSN_FUNCTION_TABLE

The DSN_FUNCTION_TABLE can be populated with information about how user-defined functions are resolved that are referred to in the SQL statement. The useful information in here includes the FUNCTION_NAME, FUNCTION_TYPE and FUNCTION_TEXT. This is information simply about the function invoked in the Explained statement, not how the function is predicted to perform.

Table 3 DSN_FUNCTION_TABLE

Column Name	Description
QUERYNO	A number intended to identify the statement being explained. If QUERYNO is not unique, the value of EXPLAIN_TIME is unique.
APPLNAME	The name of the application plan for the row, or blank.
PROGNAME	The name of the program or package containing the statement being explained, or blank.
COLLID	The collection ID for the package, or blank.
GROUP_MEMBER	The member name of the DB2 that executed EXPLAIN, or blank.
EXPLAIN_TIME	The time at which the statement is processed. This time is the same as the BIND_TIME column in PLAN_TABLE.
SCHEMA_NAME	The schema name of the function invoked in the explained statement.
FUNCTION_NAME	The name of the function invoked in the explained statement.
SPEC_FUNC_ID	The specific name of the function invoked in the explained statement.
FUNCTION_TYPE	The type of function invoked in the explained statement. Possible values: SU = Scalar function TU = Table function
VIEW_CREATOR	If the function specified in the FUNCTION_NAME column is referenced in a view definition, the creator of the view. Otherwise, blank.
VIEW_NAME	If the function specified in the FUNCTION_NAME column is referenced in a view definition, the name of the view. Otherwise, blank.

Table 3 DSN_FUNCTION_TABLE

Column Name	Description
PATH	The value of the SQL path that was used to resolve the schema name of the function.
FUNCTION_TEXT	The text of the function reference (the function name and parameters). If the function reference is over 1500 bytes, this column contains the first 1500 bytes. For functions specified in fixed notation, FUNCTION_TEXT contains only the function name. For example, for a function named /, which overloads the SQL divide operator, if the function reference is A/B, FUNCTION_TEXT contains only /, not A/B.

Additional Explain Tables

As of V8/V9 there are 11 additional Explain tables with more information to help with explaining queries. These tables are primarily for use by the IBM tools – Visual Explain and Optimization Service Center. Those products can install these tables on DB2 z/OS and then query them when you perform an Explain either through the tool, or anywhere an Explain is executed (i.e. SPUFI, QMF, etc.). Even though the intent is for use by the tools to populate information about the access paths, we can also query these tables via SQL. The tables can also be created without the tools and the DDL for them can be found on WWW.DB2EXPERT.COM, so even if you do not have the tools installed you can still create and populate these tables.

These tables are listed below:

- DSN_VIEWREF_TABLE
- DSN_QUERY_TABLE
- DSN_PGRANGE_TABLE
- DSN_SORTKEY_TABLE
- DSN_SORT_TABLE
- DSN_DETCOST_TABLE
- DSN_FILTER_TABLE
- DSN_PTASK_TABLE
- DSN_PGROUP_TABLE
- DSN_STRUCT_TABLE
- DSN_PREDICAT_TABLE

All of these tables contain very valuable information; however, for this paper we will focus on the three we find most useful for the majority of query tuning efforts.: DSN_FILTER_TABLE, DSN_PREDICAT_TABLE and DSN_DETCOST_TABLE.

DSN_FILTER_TABLE

The DSN_FILTER_TABLE contains information about how predicates are used during query processing. Some very useful columns in here are the QUERY_NO, the PREDNO and the STAGE.

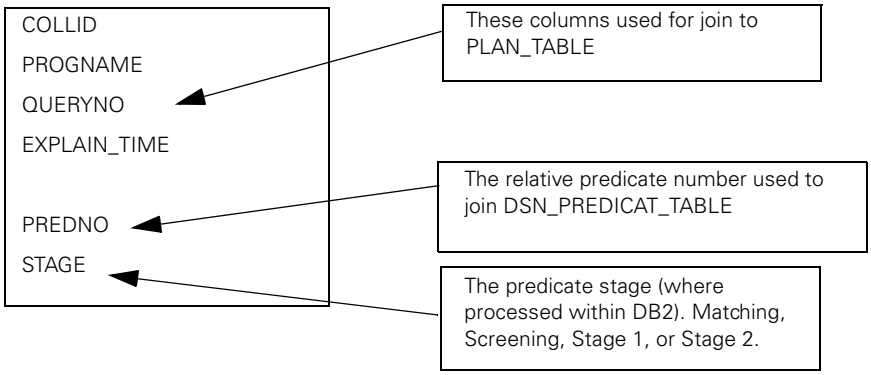
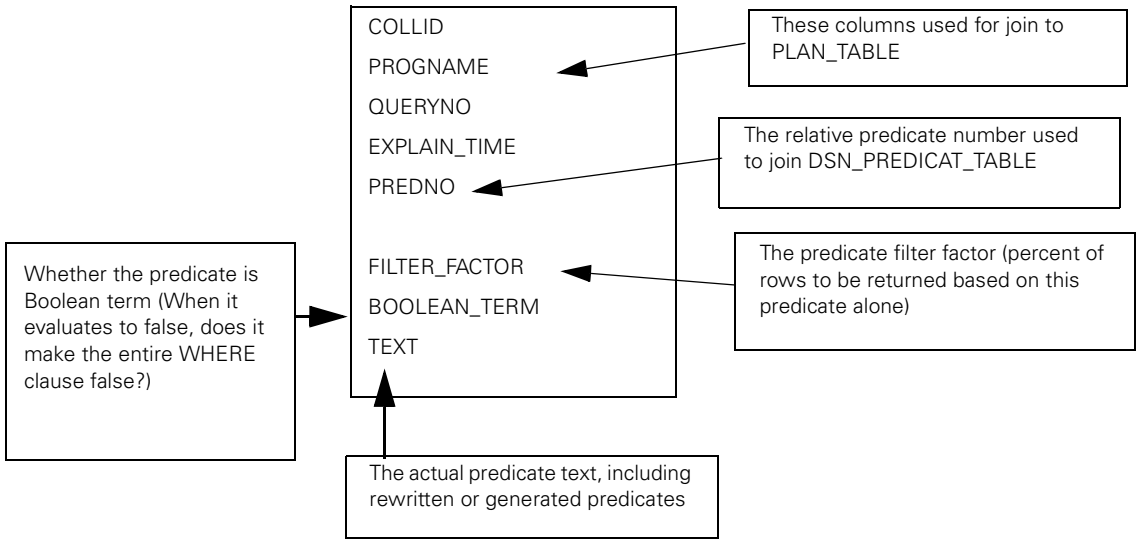


Table 4 DSN_FILTER_TABLE

Column Name	Description
QUERYNO	A number used to help identify the query being explained. It is not a unique identifier. Using a negative number will cause problems. The possible sources are: > The statement line number in the program > The QUERYNO clause > The EXPLAIN statement > The EDM unique token in the statement cache
QBLOCKNO	A number used to identify each query block within a query.
PLANNO	A number used to identify each mini-plan within a query block.
APPLNAME	The application plan name.
PROGNAME	The program name (binding an application) or the package name (binding a package).
COLLID	The collection ID for the package.
ORDERNO	The sequence number of evaluation. Indicates the order in which the predicate is applied within each stage.
PREDNO	A number used to identify a predicate within a query.
STAGE	Indicates at which stage the predicate is evaluated. The possible values are: > Matching > Screening > Stage 1 > Stage 2
ORDER_CLASS	IBM internal use only.
EXPLAIN_TIME	The EXPLAIN timestamp.
MIXOPSEQ	IBM internal use only.
REEVAL	IBM internal use only.
GROUP_MEMBER	The member name of the DB2 subsystem that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.

DSN_PREDICAT_TABLE

The DSN_PREDICATE_TABLE contains information about all the predicates in a query. Some useful columns here are the PREDNO, FILTER_FACTOR, BOOLEAN_TERM and the TEXT column. Note that now for the first time on DB2 z/OS we can see the predicates for the queries and can see the rewritten query if the optimizer has chosen to rewrite the predicates.



The following shows the columns and descriptions of the DSN_PREDICATE_TABLE (minus IBM use only columns).

Table 5 DSN_PREDICATE_TABLE

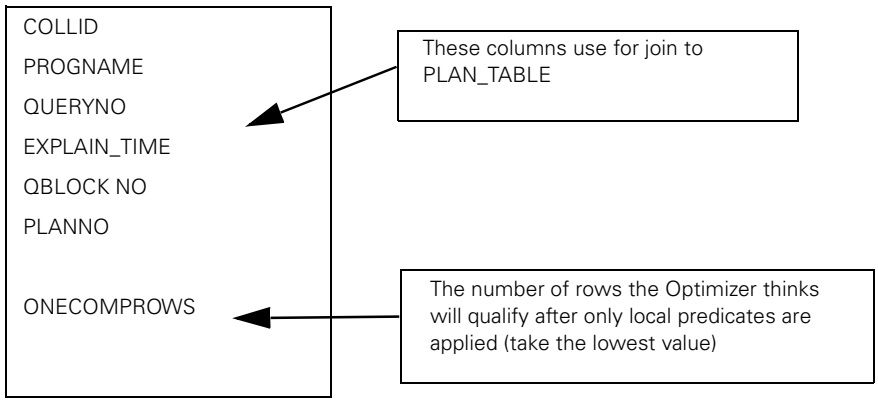
Column Name	Description
QUERYNO	A number used to help identify the query being explained. It is not a unique identifier. Using a negative number will cause problems. The possible sources are: > The statement line number in the program > The QUERYNO clause > The EXPLAIN statement > The EDM unique token in the statement cache
QBLOCKNO	A number used to identify each query block within a query.
APPLNAME	The application plan name.
PROGNAME	The program name (binding an application) or the package name (binding a package).
PREDNO	A number used to identify a predicate within a query.
TYPE	A string used to indicate the type or the operation of the predicate. The possible values are: AND OR EQUAL RANGE BETWEEN IN LIKE NOT LIKE EXISTS NOTEXIST SUBQUERY HAVING OTHERS
LEFT_HAND_SIDE	If the left-hand side (LHS) of the predicate is a table column (LHS_TABNO > 0, this column indicates the column name. Other possible values are: VALUE COLEXP NONCOLEXP CORSUB NONCORSUB SUBQUERY EXPRESSION Blanks
LEFT_HAND_PNO	
LHS_TABNO	If the LHS of the predicate is a table column, this column indicates a number that uniquely identifies the corresponding table reference within a query.
LHS_QBNO	If the LHS of the predicate is a table column, this column indicates a number that uniquely identifies the corresponding table reference within a query.
RIGHT_HAND_SIDE	If the right-hand side (RHS) of the predicate is a table column (RHS_TABNO > 0), this column indicates the column name. Other possible values are: VALUE COLEXP NONCOLEXP CORSUB NONCORSUB SUBQUERY EXPRESSION Blanks
RIGHT_HAND_PNO	If the predicate is a compound predicate (AND/OR), this column indicates the second child predicate. However, this column is not reliable when the predicate tree consolidation happens.
RHS_TABNO	If the RHS of the predicate is a table column, this column indicates a number that uniquely identifies the corresponding table reference within a query.
RHS_QBNO	If the RHS of the predicate is a subquery, this column indicates a number that uniquely identifies the corresponding query block within a query.
FILTER_FACTOR	The estimated filter factor.
BOOLEAN_TERM	Whether this predicate can be used to determine the truth value of the whole WHERE clause.
SEARCHARG	Whether this predicate can be processed by data manager (DM) stage 1. If it cannot, the relational data service (RDS) stage 2 needs to be used to take care of it, which is more costly.

Table 5 DSN_PREDICATE_TABLE

Column Name	Description
AFTER_JOIN	Indicates the predicate evaluation phase: A = After join D = During join Blank = Not applicable
ADDED_PRED	Whether the predicate is generated by transitive closure, which means DB2 can generate additional predicates to provide more information for access path selection, when the set of predicates that belong to a query logically imply other predicates.
REDUNDANT_PRED	Whether the predicate is a redundant predicate, which means evaluation of other predicates in the query already determines the result that the predicate provides.
DIRECT_ACCESS	Whether the predicate is direct access, which means one can navigate directly to the row through ROWID.
KEYFIELD	Whether the predicate includes the index key column of the involved table.
EXPLAIN_TIME	The EXPLAIN timestamp.
MARKER	Whether the predicate includes host variables, parameter markers, or special registers.
PARENT_PNO	The parent predicate number. If this predicate is a root predicate within a query block, this column is 0.
NEGATION	Whether the predicate is negated via NOT.
LITERALS	The literal value or literal values separated by colon symbols.
CLAUSE	The clause where the predicate exists: HAVING = HAVING clause ON = ON clause WHERE = WHERE clause
GROUP_MEMBER	The member name of the DB2 that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.

DSN_DETCOST_TABLE

The DSN_DETCOST_TABLE contains information about detailed cost estimation of the mini-plans in a query. The information we find most useful here is the ONECOMPROWS field. There can be multiple rows per query block so we take the one with the smallest value.



The following shows the columns and descriptions for the DSN_DETCOST_TABLE (minus IBM internal use columns).

Table 6 DSN_DETCOST_TABLE

Column Name	Description
QUERYNO	A number used to help identify the query being explained. It is not a unique identifier. Using a negative number will cause problems. The possible sources are: •The statement line number in the program •The QUERYNO clause •The EXPLAIN statement •The EDM unique token in the statement cache
QBLOCKNO	A number used to identify each query block within a query.
PLANNO	A number used to identify each mini-plan within a query block.
APPLNAME	The application plan name.
PROGNAME	The program name (binding an application) or the package name (binding a package).
OPENIO	The Do-at-open I/O cost for the non-correlated subquery.
OPENCPU	The Do-at-open CPU cost for the non-correlated subquery.
OPENCOST	The Do-at-open total cost for the non-correlated subquery.
IMFF	The filter factor of matching predicates only.
IMFFADJ	The filter factor of matching and screening predicates.
ONECOMPROWS	The number of rows qualified after applying local predicates.
DMROWS	The number of data manager rows returned (after all stage 1 predicates are applied).
DMCOLS	The number of data manager columns.
RDSROW	The number of RDS rows returned (after all stage 1 and stage 2 predicates are applied).
SNCOLS	The number of columns as sort input for a new table.
SNROWS	The number of rows as sort input for a new table.
SNRUNS	The number of runs generated for a sort of a new table.
SNMERGES	The number of merges needed during a sort.
SNCCOLS	The number of columns as sort input for a composite table.
SCROWS	The number of rows as sort input for a composite table.
SCRECSZ	The record size for a composite table.
SCPAGES	The page size for a composite table.
SCRUNS	The number of runs generated during the sort of a composite table.
SCMERGES	The number of merges needed during a sort of a composite table.
SCIOCOST	IBM internal use only.
COMPCARD	The total composite cardinality.
COMPCOST	The total cost.
EXPLAIN_TIME	The EXPLAIN timestamp.
GROUP_MEMBER	The member name of the DB2 subsystem that executed EXPLAIN. The column is blank if the DB2 subsystem was not in a data sharing environment when EXPLAIN was executed.

New Explain Query

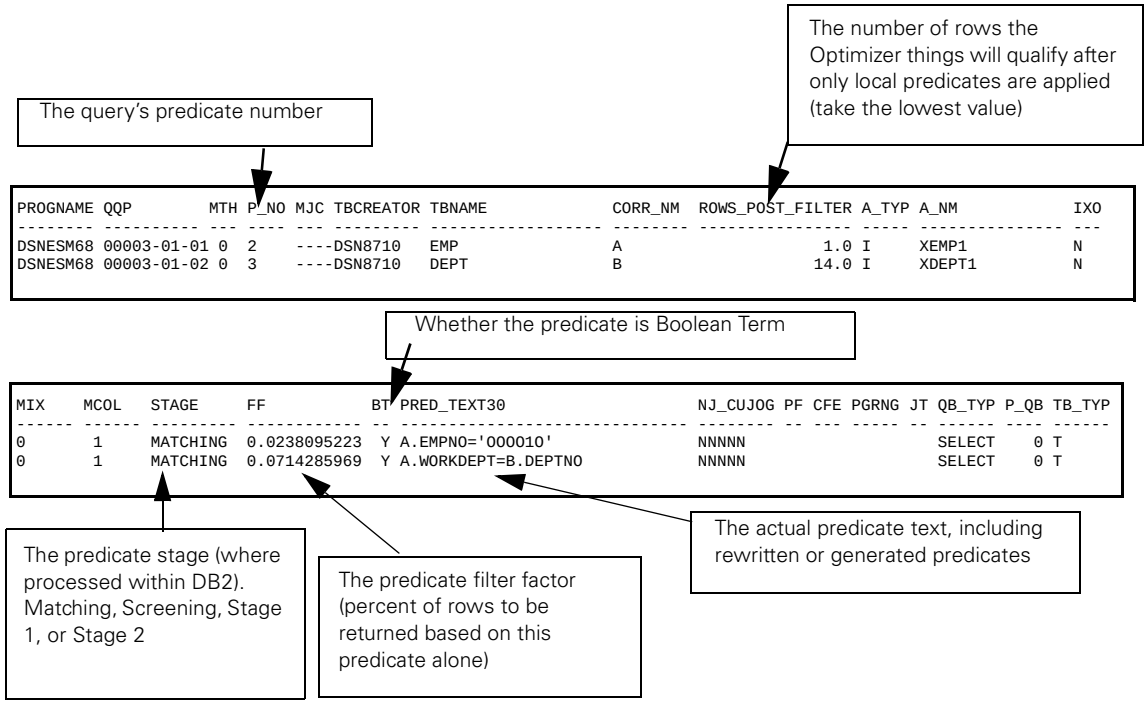
Now that there is more information available to us with the new Explain tables we need to take advantage of it and you can do so by updating the old query against the PLAN_TABLE to include some of the new information in the additional tables. The following is an example of an updated PLAN_TABLE query to add in information from the three aforementioned tables. This query can also be downloaded from WWW.DB2EXPERT.COM.

```
WITH MAXTIME (COLLID, PROGNAME, QUERYNO, BIND_TIME) AS (SELECT COLLID, PROGNAME, QUERYNO,
MAX(BIND_TIME) FROM   PLAN_TABLE WHERE  COLLID   = 'DSNESPCS' AND PROGNAME = 'DSNESM68'
GROUP  BY COLLID, PROGNAME, QUERYNO)
```

```
SELECT SUBSTR(P.PROGNAME, 1, 8) AS PROGNAME,SUBSTR(DIGITS(P.QUERYNO), 6) CONCAT '-' CONCAT
SUBSTR(DIGITS(P.QBLOCKNO), 4) CONCAT '-' CONCAT SUBSTR(DIGITS(P.PLANNO), 4) AS
QQP,SUBSTR(CHAR(P.METHOD), 1, 3) AS MTH       ,SUBSTR(CHAR(F.PREDNO), 1, 3) AS
P_NO,SUBSTR(CHAR(P.MERGE_JOIN_COLS), 1, 3) AS MJC,SUBSTR(P.CREATOR, 1, 8) AS
TBCREATOR,SUBSTR(P.TNAME, 1, 18) AS TBNAME,SUBSTR(P.CORRELATION_NAME, 1, 8) AS
CORR_NM,DEC(D.ONECOMPROWS, 10, 1) AS ROWS_POST_FILTER,P.ACCESSTYPE AS
ATYP,SUBSTR(P.ACCESSNAME, 1, 15) AS A_NM,P.INDEXONLY AS IXO,CHAR(P.MIXOPSEQ) AS MIX
,CHAR(P.MATCHCOLS) MCOL        ,F.STAGE AS STAGE,DEC(E.FILTER_FACTOR, 11, 10) AS
FF,E.BOOLEAN_TERM AS BT       ,SUBSTR(E.TEXT, 1, 30) AS PRED_TEXT30,P.SORTN_JOIN CONCAT
P.SORTC_UNIQ CONCAT P.SORTC_JOIN CONCAT P.SORTC_ORDERBY CONCAT P.SORTC_GROUPBY AS
NJ_CUJOG,P.PREFETCH AS PF,P.COLUMN_FN_EVAL AS CFE,P.PAGE_RANGE AS PGRNG,P.JOIN_TYPE AS JT
,P.QBLOCK_TYPE AS QB_TYP,P.PARENT_QBLOCKNO AS P_QB,P.TABLE_TYPE AS TB_TYP,P.BIND_TIME AS
B_TM
FROM   PLAN_TABLE P INNER JOIN   MAXTIME    M  ON     M.COLLID    = P.COLLID AND
M.PROGNAME=P.PROGNAME AND M.QUERYNO=P.QUERYNO AND M.BIND_TIME=P.BIND_TIME
LEFT JOIN   DSN_FILTER_TABLE F
ON     M.COLLID    = F.COLLID AND M.PROGNAME=F.PROGNAME AND M.QUERYNO=F.QUERYNO AND
P.QBLOCKNO=F.QBLOCKNO
AND    P.PLANNO    = F.PLANNO AND M.BIND_TIME=F.EXPLAIN_TIME AND P.ACCESSTYPE Not IN
('MX', 'MI', 'MU')
LEFT JOIN   DSN_PREDICAT_TABLE E
ON     F.PROGNAME  = E.PROGNAME AND F.QUERYNO=E.QUERYNO AND F.QBLOCKNO=E.QBLOCKNO
AND    F.PREDNO    = E.PREDNO AND M.BIND_TIME=E.EXPLAIN_TIME
LEFT JOIN TABLE  (SELECT MIN(X.ONECOMPROWS) AS ONECOMPROWS   FROM   DSN_DETCOST_TABLE X
WHERE  M.PROGNAME  = X.PROGNAME AND M.QUERYNO = X.QUERYNO AND P.QBLOCKNO = X.QBLOCKNO
AND    P.PLANNO    = X.PLANNO AND M.BIND_TIME = X.EXPLAIN_TIME) AS D ON 1=1 ORDER  BY
PROGNAME, B_TM, QQP, MIX, F.PREDNO;
```

The following is the result of the new Explain query. Note the new information we have collected with the new tables. There is some very useful information here for each predicate, including the predicate text, the filter factor, whether or not it was a Boolean term predicate, and the stage at which it will be evaluated.

Figure 1 Advanced Explain Query



Explaining Dynamic SQL

As of Version 8, there are many new ways to obtain information about your dynamic SQL that exist in the dynamic SQL cache. This gives us great opportunities for better understanding our dynamic SQL both in terms of its access paths and its run-time usage.

Explaining Access Paths

As of Version 8, you can Explain the dynamic SQL cache (global prepare cache). By using either of the following statements you can Explain the dynamic SQL cache.

```
EXPLAIN STMTSCACHE STMTID = int
```

Or

```
EXPLAIN STMTSCACHE STMTTOKEN = string
```

The access path used by the statement is written to your PLAN_TABLE with a COLLID of DSNDYNAMICSQLCACHE. The statement does NOT go through access path selection (as opposed to normal explain processing of dynamic SQL). You will need to provide the Statement Id (STMTID) available from trace records with IFCID 316, 124 or the statement token (STMTTOKEN) assigned by application that prepares statement which is provided by the RRSAF SET_ID function, or the SQLESETI function for remote applications.

DSN_STATEMENT_CACHE_TABLE

There is also a new table in Version 8 allowing you to populate it by using the keyword ALL is on EXPLAIN STMTCACHE statement. This will populate the DSN_STATEMENT_CACHE_TABLE. This table is designed to hold the output of IFCID 316 and IFCID 318 for the execution information of the SQL statements in the dynamic SQL cache.

The contents of the rows show identifying information about the cached entries as well as an accumulation of statistics reflecting the executions of the statements by all processes that have executed the statement. The information in the DSN_STATEMENT_CACHE_TABLE is nearly identical to the information returned from the IFI monitor READS API for IFCIDs 0316 and 0317. Note that the collection and reset of the statistics in these records is controlled by starting and stopping IFCID 318.

The population of this table is done by issuing the following statement:

```
EXPLAIN STMTCACHE ALL
```

The invocation of the state will populate the table with information about all of the statements in the dynamic statement cache associated with your SQLID. However, if you have SYSADM authority then all cached statements will be exposed.

The DSN_STATEMENT_CACHE_TABLE must be created first before it can be populated. This can be done with through the IBM Visual Explain or Optimization Service Center or by executing the DDL documented in the IBM Administration Guide.

The following describes the columns in the DSN_STATEMENT_CACHE_TABLE.

Table 7 DSN_STATEMENT_CACHE_TABLE

Column Name	Description
STMT_ID	An EDM unique token.
STMT_TOKEN	A user-provided identification string.
COLLID	Collection ID; value is DSNDYNAMICSQLCACHE.
PROGRAM_NAME	Name of package or DBRM that performed the initial PREPARE.
INV_DROPALT	Invalidated by DROP/ALTER.
INV_REVOKE	Invalidated by REVOKE.
INV_LRU	Removed from cache by LRU.

Table 7 DSN_STATEMENT_CACHE_TABLE

Column Name	Description
INV_RUNSTATS	Invalidated by RUNSTATS.
CACHED_TS	Timestamp when statement was cached.
USERS	Number of current users of statement. These are the users that have prepared or executed the statement during their current unit of work.
COPIES	Number of copies of statement owned by all threads in the system.
LINES	Precompiler line number from the initial PREPARE.
PRIMAUTH	Primary authorization ID of the user that did the initial PREPARE.
CURSQLID	CURRENT SQLID of the user that did the initial prepare.
BIND_QUALIFIER	Bind object qualifier for unqualified table names.
BIND_ISO	ISOLATION bind option: UR = Uncommitted read CS = Cursor stability RS = Read stability RR =Repeatable read
BIND_CDATA	DATA CURRENTDATA bind option: Y = CURRENTDATA(YES) N = CURRENTDATA(NO)
BIND_DYNRL	DYNAMICRULES bind option: B = DYNAMICRULES(BIND) R = DYNAMICRULES(RUN)
BIND_DEGRE	CURRENT DEGREE value: A = ANY 1 = 1
BIND_SQLRL	CURRENT RULES value: D = DB2 S = SQL
BIND_CHOLD	Cursor WITH HOLD bind option: Y = Initial PREPARE was done for a cursor WITH HOLD N = Initial PREPARE was not done for a cursor WITH HOLD
STAT_TS	Timestamp of stats when IFCID 318 is started.
STAT_EXEC	Number of executions of statement. For a cursor statement, this value is the number of OPENs.
STAT_GPAG	Number of getpage operations performed for statement.
STAT_SYNR	Number of synchronous buffer reads performed for statement.
STAT_WRIT	Number of buffer write operations performed for statement.
STAT_EROW	Number of rows examined for statement.
STAT_PROW	Number of rows processed for statement.
STAT_SORT	Number of sorts performed for statement.
STAT_INDX	Number of index scans performed for statement.
STAT_RSCN	Number of table space scans performed for statement.
STAT_PGRP	Number of parallel groups created for statement.
STAT_ELAP	Accumulated elapsed time used for statement.
STAT_CPU	Accumulated CPU time used for statement.
STAT_SUS_SYNIO	Accumulated wait time for synchronous I/O.
STAT_SUS_LOCK	Accumulated wait time for lock and latch requests.
STAT_SUS_SWIT	Accumulated wait time for synchronous execution unit switch.
STAT_SUS_GLCK	Accumulated wait time for global locks.
STAT_SUS_OTHR	Accumulated wait time for read activity done by another thread.
STAT_SUS_OTHW	Accumulated wait time for write activity done by another thread.
STAT_RIDLIMT	Number of times a RID list wasn't used because the number of RIDs would have exceeded one or more DB2 limits.

Table 7 DSN_STATEMENT_CACHE_TABLE

Column Name	Description
STAT_RIDSTOR	Number of times a RID list wasn't used because not enough storage was available to hold the list of RIDs.
EXPLAIN_TS	When the statement cache table is populated.
SCHEMA	CURRENT SCHEMA value.
STMT_TEXT	Statement text.
STMT_ROWID	Statement ROWID.
BIND_RA_TOT	The total number of REBIND commands that have been issued for the dynamic statement because of the REOPT(AUTO) option.
BIND_RO_TYPE	The current specification of the REOPT option for the statement: N = REOPT(NONE) 1 = REOPT(ONCE) or its equivalent A = REOPT(AUTO) or its equivalent 0 = The current plan is deemed optimal and there is no need for REOPT(AUTO)

Note that the information populated here is very different from the type of information you populate in the PLAN_TABLE. The PLAN_TABLE contains information about the access path. Here we have information about the executions of the statements in the dynamic SQL cache.

So now you have two types of information that you can obtain for dynamic SQL in the cache via Explain:

- EXPLAIN STMTCACHE with the STMTID or STMTTOKEN

  Provides traditional access path information to be written to the PLAN_TABLE for the associated SQL statement with a single row written to DSN_STATEMENT_CACHE_TABLE if it exists.

- STMTCACHE with the ALL keyword

  Populates the DSN_STATEMENT_CACHE_TABLE which consists of one row per SQL statement in the dynamic statement cache for which the current authorization ID is authorized to execute. Note that the STMTID and STMTTOKEN are provided here and then they can be used with the EXPLAIN STMTCACHE with STMTID or STMTTOKEN to obtain the access path of the statement.

Using Explain Output for Rebind Evaluation

Rebinds of applications can be a very risky action due to the fact that access paths can change and sometimes not for the better. Rebinds are necessary due to application changes or possibly a release migration in order to pick up improved optimization.

To mitigate the risk of an access path going bad we use Explain information from the PLAN_TABLE to either evaluate access paths after a rebind and/or to be able to preserve an access path by applying an optimization hint to regain our old access paths.

One method to test the effects of new release code to see access path enhancements includes the following steps:

- REBIND all plans under a different name using EXPLAIN(YES)
- Perform a BIND PACKAGE(*) EXPLAIN(YES) into a separate collection
- Compare old and new access paths

If major changes occur optimization hints may be an option to restore old access path.

Because it is recommended both in version 8 and 9 that rebinds be performed to pick up improved optimization, having an Explain done on all critical packages before a DB2 release migration is a good idea so that the access paths are preserved and if an access path changes, the old path maybe regained via a hint. But if you have not collected this information and kept it, it is hard to know exactly what changed and how to fix it.

Using Explain Output to Tune Queries

The following example shows the result of adding a redundant Boolean term predicate to a query. Query 1 contains the originally scrolling cursor, and query 2 the improved cursor with a redundant predicate. The EXPLAIN query output demonstrates the improvement.

Query 1

```
EXPLAIN PLAN SET QUERYNO=1 FOR
SELECT *
FROM   DSN8710.EMP
WHERE  EMPNO = ?
OR     (EMPNO = ? AND
        LASTNAME >= ?)
OR     EMPNO > ?
ORDER BY EMPNO, LASTNAME;
```

Query 2

```
EXPLAIN PLAN SET QUERYNO=2 FOR
SELECT *
FROM   DSN8710.EMP
WHERE  (EMPNO = ?
OR     (EMPNO = ? AND
        LASTNAME >= ?)
OR     EMPNO > ?)
AND    EMPNO >= ?
ORDER BY EMPNO, LASTNAME;
```

While the two queries, when executed with the same input parameters, will produce the same results, the second query introduces a redundant Boolean term predicate that improves indexability. While the PLAN_TABLE shows the improved index access it really doesn't explain why. By querying the additional Explain tables we can see the predicates, and the explanation of the improvement.

Here is the enhanced Explain query output for the two queries above:

Figure 2 Advanced Explain for Query 1

```
PROGNAME QQP        MTH P_NO MJC TBCREATOR TBNAME             CORR_NM  ROWS_POST_FILTER A_TYP A_NM            IXO
-------- ---------- --- ---- --- --------- ------------------ -------- ---------------- ----- --------------- ---
DSNESM68 00001-01-01 0  2    --- DSN8710   EMP                --------             15.3 I     XEMP1           N
DSNESM68 00001-01-01 0  4    --- DSN8710   EMP                --------             15.3 I     XEMP1           N
DSNESM68 00001-01-01 0  5    --- DSN8710   EMP                --------             15.3 I     XEMP1           N
DSNESM68 00001-01-01 0  6    --- DSN8710   EMP                --------             15.3 I     XEMP1           N
```

```
MIX    MCOL   STAGE     FF           BT PRED_TEXT30                    NJ_CUJOG PF CFE PGRNG JT QB_TYP P_QB TB_TYP
------ ------ --------- ------------ -- ------------------------------ -------- -- --- ----- -- ------ ---- ------
0       0     STAGE1    .0238095223  N  DSN8710.EMP.EMPNO=(EXPR)       NNNNN                    SELECT    0 T
0       0     STAGE1    .0238095223  N  DSN8710.EMP.EMPNO=(EXPR)       NNNNN                    SELECT    0 T
0       0     STAGE1    .3333333134  N  DSN8710.EMP.LASTNAME>=(EXPR)   NNNNN                    SELECT    0 T
0       0     STAGE1    .3333333134  N  DSN8710.EMP.EMPNO>(EXPR)       NNNNN                    SELECT    0 T
```

Figure 3 Advanced Explain for Query 2

```
PROGNAME QQP        MTH P_NO MJC TBCREATOR TBNAME             CORR_NM  ROWS_POST_FILTER A_TYP A_NM            IXO
-------- ---------- --- ---- --- --------- ------------------ -------- ---------------- ----- --------------- ---
DSNESM68 00002-01-01 0  3    --- DSN8710   EMP                --------              5.1 I     XEMP1           N
DSNESM68 00002-01-01 0  5    --- DSN8710   EMP                --------              5.1 I     XEMP1           N
DSNESM68 00002-01-01 0  6    --- DSN8710   EMP                --------              5.1 I     XEMP1           N
DSNESM68 00002-01-01 0  7    --- DSN8710   EMP                --------              5.1 I     XEMP1           N
DSNESM68 00002-01-01 0  8    --- DSN8710   EMP                --------              5.1 I     XEMP1           N
```

```
MIX    MCOL   STAGE     FF           BT PRED_TEXT30                    NJ_CUJOG PF CFE PGRNG JT QB_TYP P_QB TB_TYP
------ ------ --------- ------------ -- -------------------------------- -------- -- --- ----- -- ------ ---- ------
0       1     STAGE1    .0238095223  N  DSN8710.EMP.EMPNO=(EXPR)        NNNNN                    SELECT    0 T
0       1     STAGE1    .0238095223  N  DSN8710.EMP.EMPNO=(EXPR)        NNNNN                    SELECT    0 T
0       1     STAGE1    .3333333134  N  DSN8710.EMP.LASTNAME>=(EXPR)   NNNNN                    SELECT    0 T
0       1     STAGE1    .3333333134  N  DSN8710.EMP.EMPNO>(EXPR)        NNNNN                    SELECT    0 T
0       1     MATCHING  .3333333134  Y  DSN8710.EMP.EMPNO>(EXPR)        NNNNN                    SELECT    0 T
```

As indicated in the last line of the output for the Explain of query 2, the additional redundant predicate is the only Boolean term predicate, and so is the only predicate that matched on an index. So, the advanced Explain output not only tells us that the access path improved, but also why it improved.

Conclusion

The Explain information available in both version 8 and 9 is far more detailed than in past releases. We need to look at the new information available and develop methods for using it. Whether these methods are home grown or through the use of tools, we can miss out on this valuable resource. Our query performance is depending on it!

References

The following IBM manuals were used as references

- IBM DB2 9 for z/OS SQL Reference - SC18-9854-00
- IBM DB2 9 for z/OS Performance Monitoring and Tuning Guide - SC18-9851-00
- IBM DB2 9 for z/OS Administration Guide - SC18-9840-00

BMC SQL Performance for DB2

BMC SQL Performance for DB2 is an integrated solution that provides comprehensive capabilities for managing SQL and application performance across the application life cycle, from development to production. The solution includes BMC APPTUNE for DB2, BMC SQL Explorer for DB2, and the BMC Index Component, which is available only in the solution.

In production, BMC APPTUNE for DB2 captures and analyzes detailed execution metrics, identifying problem SQL statement and offering recommendations for improvements.

BMC SQL Explorer for DB2 provides an extended version of the Explain discussed in this paper. BMC SQL Explorer for DB2 looks at static and dynamic SQL. The SQL can be captured from the DB2 catalog, dynamic SQL text from production workloads, or entered directly via the user interface. The SQL text is explained and analyzed, providing details on any issues found for each query block and the overall statement as well. For example, an access path using list prefetch is identified for further evaluation. Groups of SQL statements can be analyzed in real time or batch mode, providing an easy way to evaluate all statements for a given package, plan, or application.

Explain data is captured and stored for historical analysis over time. This analysis can identify changes in SQL access paths that may increase cost and degrade performance as you migrate the application to a new DB2 version. This very problem occurred in the migration from DB2 V7 to DB2 V8. Identifying these issues early can help avoid a negative impact to production performance. You can integrate this step into the application promotion procedure: add a BMC SQL Explorer for DB2 step to the procedure to compare the current cost of the SQL statement with the new cost as determined by the Explain. If the cost increases by a factor you set, the job step can return a condition code that will not allow the subsequent bind processes to complete.

The BMC SQL Performance for DB2 Index Component provides specific functions for evaluating your indexing strategies. The Index Component help you identify SQL statements using poor indexing options to satisfy the requests. You can than use the “what-if” capability to evaluate other indexes that can improve performance. You can identify indexing problems at the DB2 object level. You can identify indexes that are not being used for read access during specific time intervals to see which indexes are being used minimally and to find a starting point for further analysis.

About BMC Software

BMC Software delivers the solutions IT needs to increase business value through better management of technology and IT processes. Our industry-leading Business Service Management solutions help you reduce cost, lower the risk of business disruption, and benefit from an IT infrastructure built to support business growth and flexibility. Only BMC provides best-practice IT processes, automated technology management, and award-winning BMC Atrium technologies that offer a shared view into how IT services support business priorities. Known for enterprise solutions that span mainframe, distributed systems, and end-user devices, BMC also delivers solutions that address the unique challenges of the midsized business. Founded in 1980, BMC has offices worldwide and fiscal 2007 revenues of $1.58 billion. Activate your business with the power of IT. www.bmc.com.

July 22, 2008

93567