Using the EXPLAIN Feature to Improve DB2® Query Performance

Frank Solomon, Blue Cross Blue Shield of CT
North Haven, CT 06473

ABSTRACT

This paper discusses the use of the SQL EXPLAIN feature of DB2 to assist in the understanding of how DB2 handles an SQL query request and how to look for tuning opportunities for better query performance. This paper will explain how DB2 processes an SQL request, how to use the SQL EXPLAIN statement with the SAS® SQL PASS-THROUGH® Facility, obtain the EXPLAIN results, and interpret what those results mean. In addition, this paper will present some helpful ideas for optimizing queries to save both time and system overhead.

INTRODUCTION

To acquire the skills to enhance SQL query performance it is important to understand the underlying component structures that make up the DB2 DBMS. DB2 is a Database Management System comprising of two dimensional tables and indexes that form a basic relational structure. SQL provides the means for data access and retrieval from DB2 tables. Internally, DB2 contains tables that form a catalog that store information about all DB2 objects. An intelligent component of DB2, the optimizer, determines the probable access path of a single SQL statement to the data determined by information contain in the DB2 catalog. The optimizer is an inference engine that makes all DB2 data access decisions. The optimizer’s decision can only be indirectly influenced through coding strategies in the SQL code. The SQL EXPLAIN is used to request that DB2 explain the optimizer’s access path decision. Access path performance may be affected by:

- Number of rows to be accessed
- Distribution of data
- Access strategy
- Sequencing of data
- Condition of the databases
- Requested approach to the data

Table INDEXES enhance access path performance by:

- Providing pointer controlled access to data
- Avoiding scanning of all table data
- Avoiding sorts
- Avoiding access to the table data entirely

How does DB2 process an SQL query?

The SQL statement that is presented to DB2, whether dynamically (SAS, QMF) or through the BIND process (application programs), must move through many internal DB2 components to locate and retrieve physical table data. The following gives a flow and description of the DB2 components used to satisfy an SQL query request:

SQL Parser
- Checks for proper SQL syntax

Optimizer
- Confirms the users authority
- Decomposes query into query blocks
  a query block represents a section of an SQL statement e.g., base SELECT, a subquery, a UNION
- Merges query with any DB2 table views being accessed in the query
- Obtains catalog statistics for DB2 objects accessed
- Chooses an access path:
  - Sequences query blocks
  - Computes row filter factors
  - Computes access path costs
  - Determines whether to use an index
  - Determines which indexes to use
  - Chooses least costly path
  - Creates the access plan

Code generator
- Converts access plan to object machine code
- Stores access plan as an Application Plan for statically bound applications

Executor
- Executes stored Application Plans to access data
- Execute dynamic (Ad-Hoc) queries one time only

Data Retrieval
- Data Manager-DM
- Stage 1 row filter, called sargable
  - Evaluates predicates (from WHERE clause) as the data is retrieved from the physical storage (resource efficient)
  - Uses the Buffer Manager to access the tables
- Used by the Relational Data System-RDS
- Relational Data System-RDS
- Stage 2 row filter, called non-sargable
  - Interface for queries (creates results table)
  - Makes calls to the Data Manager
  - Evaluates predicates (from WHERE clause) after the data has been retrieved in the buffer (resource intensive)
- Buffer Manager
  - Works like a scratch pad area for DB2
  - Interfaces between the VSAM Media Manager and the Data Manager
- VSAM Media Manager
  - Manages the physical data storage

How to EXPLAIN

Before any query can be explained, a plan_table prefixed by the User ID doing the explaining must be defined in the DB2 subsystem where the tables that are being accessed reside. DB2 will insert rows representing the results of issuing an SQL EXPLAIN in the userid.PLAN_TABLE. The format of the PLAN_TABLE is as follows:

```sql
CREATE TABLE USERID.PLAN_TABLE

 QuéryNO INTEGER NOT NULL,
 OBLOCKNO SMALLINT NOT NULL,
 APPLNAME CHAR(8) NOT NULL,
 PROGRAM CHAR(8) NOT NULL,
 PLANNO SMALLINT NOT NULL,
 METHOD SMALLINT NOT NULL,
 CREATOR CHAR(8) NOT NULL,
 TNAME CHAR(18) NOT NULL,
 TABNO SMALLINT NOT NULL,
 ACCESTYPE CHAR(2) NOT NULL,
 MATCHCOLS SMALLINT NOT NULL,
 ACCESSCREATOR CHAR(8) NOT NULL,
 ACCESSNAME CHAR(18) NOT NULL,
 INDEXONLY CHAR(1) NOT NULL,
 SORTN_UNIQUE CHAR(1) NOT NULL,
 SORTN_JOIN CHAR(1) NOT NULL,
 SORTN_ORDERBY CHAR(1) NOT NULL,
 SORTN_GROUPLBY CHAR(1) NOT NULL,
 EXPLAIN_UNIQUE CHAR(1) NOT NULL,
 EXPLAIN_JOIN CHAR(1) NOT NULL,
 EXPLAIN_ORDERBY CHAR(1) NOT NULL,
 EXPLAIN_GROUPLBY CHAR(1) NOT NULL,
 STLOCKMODE CHAR(3) NOT NULL,
 TIMESTAMP CHAR(16) NOT NULL,
 REMARKS VARCHAR(254) NOT NULL,
 PREFETCH CHAR(1) NOT NULL WITH DEFAULT,
 COLUMN_ENUM CHAR(1) NOT NULL WITH DEFAULT,
 MIXEDSEQ SMALLINT NOT NULL WITH DEFAULT,
 VERSION VARCHAR(64) NOT NULL WITH DEFAULT,
 COLLID CHAR(18) NOT NULL WITH DEFAULT

IN DBNAME.TNAME;
```

The following is a simple coding sample to illustrate issuing an SQL EXPLAIN for a query:

```sql
SELECT using the SQL PASS-THROUGH Facility:

OPTIONS LINESIZE=250 PAGESIZE=60;
PROC SQL:
CONNECT TO DB2 (SSID=yourdb2system):
EXECUTE (EXPLAIN ALL SET QUERIN=1 FOR SELECT A.COL1
 .A.COL2
 .B.COL3
 .B.COL4
FROM TABLEA A
 .TABLEB B
WHERE A.COL1 IN (?)
AND A.COL2 BETWEEN ? AND ?
AND B.COL3 = ?
AND A.COL1 = B.COL1
AND A.COL2 = B.COL2
) BY DB2;

INPUT SQLXML:
TITLE "DB2 Query Explain Results":
CREATE TABLE TEXPLAIN AS SELECT *
FROM CONNECTION TO DB2 (SELECT *
FROM userid.PLAN_TABLE
WHERE QUERIN = 1):

INPUT SQLXML:
DISCONNECT FROM DB2:
RUN:
PROC PRINT DATA=TEXPLAIN:
RUN:

197
To explain a query, the only element that is required is that the statement \texttt{EXPLAIN ALL SET QUERYNO = 1} must precede the \texttt{SELECT} clause. To issue an \texttt{EXPLAIN}, the \texttt{EXECUTE} statement (enables DBMS specific SQL) must be used or an error will occur. The \texttt{EXECUTE EXPLAIN} statement can be commented out when ready to actually execute the query. When executing these statements, DB2 will insert rows representing the \texttt{EXPLAIN} data in the \texttt{PLAN_TABLE}. The query explain rows will be identified by the number that is used in the SET \texttt{QUERYNO = n...} so that this particular query can be easily located when querying the \texttt{PLAN_TABLE}. Notice that "?" (parameter markers) are used in place of actual data. Using actual data may cause a different access path than using "?" due to different filter factors. Use "?" when the data is unknown or will vary to obtain the DB2 default factor. Use actual data if it is known or static to obtain the actual filter factor results.

A filter factor of a \texttt{WHERE} predicate is a percentage that estimates how many rows are rejected by and how many rows are not rejected by the \texttt{WHERE} predicate. Filter factors greatly affect the estimated costs of the DB2 access paths to be chosen by the optimizer by estimating the percentage of the rows returned after evaluating the \texttt{WHERE} predicate. Use the \texttt{&SQLMSG} macro variable which contains the return code and error descriptions produced by DB2 to check for successful execution of the SQL statements. Appendix A shows an example of some explained output from a \texttt{PLAN_TABLE}.

What do the \texttt{EXPLAIN} columns mean?

The \texttt{PLAN_TABLE} columns that relate to \texttt{GENERAL} information about the explained SQL are as follows:

- \texttt{QUERYNO} - Query number assigned by the explaining user
- \texttt{QUERYNO} - Number that identifies the SQL query sections for a query with a \texttt{SELECT}, subquery, \texttt{UNION}
- \texttt{APPLNAME} - Plan name for programs with embedded SQL (C, COBOL, etc)
- \texttt{PROGNAME} - Program name for programs with embedded SQL
- \texttt{PLANNO} - Identifies the order of operations within a \texttt{QUERYNO}
- \texttt{CREATOR} - Creator of a new table (materialized view) accessed
- \texttt{TNAME} - Name of new table (materialized view) accessed

The \texttt{PLAN_TABLE} columns that relate to \texttt{INDEX} usage information about the explained SQL are as follows:

- \texttt{TABNO} - Number that identifies the sequence of references to the same table in the \texttt{FROM} clause
- \texttt{TSLOCKMODE} - The \texttt{TABLESPACE} lock mode
  - \texttt{IS} - Intent share
  - \texttt{IX} - Intent exclusive
  - \texttt{SIX} - Share with intent exclusive
  - \texttt{S} - Share
  - \texttt{U} - Update
  - \texttt{X} - Exclusive
- \texttt{TIMESTAMP} - Date and time the \texttt{EXPLAIN} was processed
- \texttt{REMARKS} - Field in which comments can be inserted

The \texttt{PLAN_TABLE} columns that relate to \texttt{SORT} usage information about the explained SQL are as follows:

- \texttt{ACCESSTYPE} - Type of table \texttt{INDEX} usage:
  - \texttt{R} - Pull table scan (uses no index)
  - \texttt{I} - Use an index
  - \texttt{II} - One-fetch scan (MIN or MAX functions)
  - \texttt{N} - Index scan (predicate uses an IN)
  - \texttt{M} - Multi-index scan followed by:
    - \texttt{MX} - Matching index scan on row pointers only
    - \texttt{MI} - Consolidate (intersect) of row pointers from multiple indexes
    - \texttt{MU} - Combination (union) of row pointers from multiple indexes
- \texttt{MATCHCOUNTS} - Number of index keys used in an index scan
- \texttt{ACCESSCREATOR} - Creator of the index being used
- \texttt{ACCESSNAME} - Name of the index being used
- \texttt{INDEXONLY} - Y if all data comes from the index
  - N if data must come from the table
- \texttt{INDEXSEQ} - Sequence of steps in a multi-index scan

The \texttt{PLAN_TABLE} columns that relate to \texttt{SORT} usage information about the explained SQL are as follows:

- \texttt{METHOD} - Indicates the table \texttt{JOIN} method
  - \texttt{0} - First table accessed. First outer table
1 - Nested loop join
2 - Merge scan join
3 - Sort required for ORDER BY. GROUP BY. UNION. DISTINCT
4 - Hybrid join

SORTN_UNIQ - Not being used (always N)
SORTN_JOIN - Sort inner table for join processing (Y.N)
SORTN_ORDERBY - Not being used (always N)
SORTN_GROUPBY - Not being used (always N)
SORTC_UNIQ - Sort to remove duplicates (Y.N)
SORTC_JOIN - Sort outer table for join processing (Y.N)
SORTC_ORDERBY - Sort for the ORDER BY clause (Y.N)
SORTC_GROUPBY - Sort for the GROUP BY clause (Y.N)

PREFETCH - Physical data read in advance:
  S - Sequential prefetch
  L - List prefetch
  Blank - No prefetch

COLUMN_FN_EVAL - Indicates when a column function is evaluated:
  R - At data retrieval time (stage 1)
  S - At sort time (stage 2)

What should you be looking for?

Know what columns make up the indexes on the tables being accessed. Indexes enhance performance and reduce costs. Look at the ACCESSTYPE to see if an index is being used. An ACCESSTYPE of "R" means all the data in the table must be scanned and no indexes are being used. For larger tables and joins this is very expensive. Look for MATCHCOLS to see how many index keys are being used. The more the better. Check to see if INDEXONLY is "Y". A "Y" means that data will be retrieved from the index rather than the table (very fast). Matching on a unique index provides the best filtering of data. Change the WHERE clause to take advantage of indexes. Have indexes created to match WHERE clauses that are common and often.

Avoid unnecessary sorts as much as possible.

DISTINCT. UNION. ORDER BY. GROUP BY. and some joins require DB2 to sort data. DB2 is sorting if METHOD = "3" and the SORTN or SORTC columns = "Y". PREFETCH is good if lots of rows are being accessed. Prefetch is when DB2 fetches physical data into the buffer ahead of time before being analyzed. Improves performance of some types of queries by reducing the number of physical I/Os required to return large sets of data. Very effective when the table data is in clustered sequence. Could be a problem or degrade performance if expecting accessing a small amount of data.

Table JOINS support multiple table access with matching and frequently make use of primary and foreign keys. JOIN performance is contingent on indexes to support matching and WHERE predicates. When joining tables look at the METHOD column. A nested loop join is preferred if accessing a small percentage of rows or when index columns are used in the join. Nested Loop joins are favored by DB2 when the appropriate indexes exist and the result table will be small. Merge scan join is preferred when accessing a large percentage of rows or index columns are not used in the join. Merge Scan joins may require sorting prior to joining. Hybrid joins make use of list prefetch and process duplicate outer table matches most efficiently. DB2 favors a Hybrid join for medium sized result tables. Joins are more efficient than subqueries. Joins make better use of indexes than subqueries. Rewrite subqueries as joins when possible. A correlated subquery refers to an outer query. DB2 will sometimes hold the results of a correlated subquery to evaluate against each row from the outer query.

Make predicates indexable whenever possible. Indexable predicates are usually evaluated at STAGE 1 filtering through the DB2 Data Manager. STAGE 1 filtering is more efficient because data is filtered as the rows are being retrieved. The following are indexable predicates:

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLUMN = value</td>
<td>Indexable predicate.</td>
</tr>
<tr>
<td>COLUMN &gt; value (also &gt;=, not&lt;)</td>
<td>Indexable predicate.</td>
</tr>
<tr>
<td>COLUMN &lt; value (also &lt;=, not&gt;)</td>
<td>Indexable predicate.</td>
</tr>
<tr>
<td>COLUMN IS NULL</td>
<td>Indexable predicate.</td>
</tr>
<tr>
<td>COLUMN LIKE 'char_%'</td>
<td>Indexable predicate.</td>
</tr>
<tr>
<td>COLUMN IN (list)</td>
<td>Indexable predicate.</td>
</tr>
<tr>
<td>COLUMN BETWEEN value1 AND value2</td>
<td>Indexable predicate.</td>
</tr>
</tbody>
</table>

Stage 2 predicates are usually less efficient because data is filtered or evaluated after the rows have been physically retrieved. STAGE 2 predicates are evaluated last and are the costliest to process. The following are STAGE 2 predicates:

A COLUMN = B.COLUMN2 (with any condition)
COLUMN = value + 1 (any numeric calculation)
DAY(COLUMN) = 1 (any scalar functions)
COLUMN||COLUMN2 = 'value' (concatenations)

Rewrite predicates to take advantage of indexes and STAGE 1 processing. Check ACCESSTYPE column.

Some performance tips.

The following are some performance tips you might consider when constructing your SQL or to change EXPLAIN results to improve the retrieval of data.

- Make all queries as explicit as possible
- The WHERE clause is a filter, use it effectively
- Select only columns that are needed. Do not use SELECT *
- Use the IN statement instead of multiple ORs
- Qualify all join columns in a JOIN. Join with as many of the index columns as possible
- Do not use a JOIN without a qualifying WHERE clause
- Do not specify tables in a JOIN if you don't need them
- Avoid using arithmetic expressions in the WHERE clause
- Avoid using character concatenations (||) in the WHERE clause
- Avoid using SUBSTRING in the WHERE clause
- Use <= and => instead of NOT BETWEEN
- Avoid using NOT =, NOT BETWEEN, NOT IN, NOT LIKE
- Do not use an ORDER BY with DISTINCT or UNION if the order you want is the column order of the SELECT clause
- Avoid using a subquery in an IN list if the subquery returns multiple rows. Use a JOIN instead
- Use NOT EXISTS instead of NOT IN for a subquery

CONCLUSION

The key to a healthy query environment is the way in which we access and manipulate the data that is made available to us. Using the EXPLAIN feature of DB2 can help in giving insight into the way DB2 will handle the execution of the SQL query. Since the DB2 optimizer makes that decision for us, it is possible to help influence that decision by applying various techniques and re-formulating the SQL code. Understanding DB2 and the processes that take place behind the scenes can greatly enhance ones performance and promote a more efficient query environment for all to use. Make it a common practice to use the EXPLAIN statement and develop the skills to write performance oriented query requests.

REFERENCES


<table>
<thead>
<tr>
<th>QUERYNO</th>
<th>QBLOCKNO</th>
<th>APPLNAME</th>
<th>PROGRAM</th>
<th>PLANNO</th>
<th>METHOD</th>
<th>CREATOR</th>
<th>TNAME</th>
<th>TABNO</th>
<th>ACCESTYPE</th>
<th>MATCHCOLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1</td>
<td>DSQ8ESQL</td>
<td>1</td>
<td>0</td>
<td>PROD</td>
<td></td>
<td>TITCP012_CLM_VR_2</td>
<td>1</td>
<td>M</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td>DSQ8ESQL</td>
<td>1</td>
<td>0</td>
<td>PROD</td>
<td></td>
<td>TITCP012_CLM_VR_2</td>
<td>1</td>
<td>M</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td>DSQ8ESQL</td>
<td>1</td>
<td>0</td>
<td>PROD</td>
<td></td>
<td>TITCP012_CLM_VR_2</td>
<td>1</td>
<td>M</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td>DSQ8ESQL</td>
<td>1</td>
<td>0</td>
<td>PROD</td>
<td></td>
<td>TITCP012_CLM_VR_2</td>
<td>1</td>
<td>M</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>1</td>
<td>DSQ8ESQL</td>
<td>1</td>
<td>0</td>
<td>PROD</td>
<td></td>
<td>TITCP012_CLM_VR_2</td>
<td>1</td>
<td>M</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>2</td>
<td>DSQ8ESQL</td>
<td>1</td>
<td>0</td>
<td>PROD</td>
<td></td>
<td>TITCP013_CLM_VR_3</td>
<td>1</td>
<td>M</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>2</td>
<td>DSQ8ESQL</td>
<td>1</td>
<td>0</td>
<td>PROD</td>
<td></td>
<td>TITCP013_CLM_VR_3</td>
<td>1</td>
<td>M</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>2</td>
<td>DSQ8ESQL</td>
<td>1</td>
<td>0</td>
<td>PROD</td>
<td></td>
<td>TITCP013_CLM_VR_3</td>
<td>1</td>
<td>M</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>2</td>
<td>DSQ8ESQL</td>
<td>1</td>
<td>0</td>
<td>PROD</td>
<td></td>
<td>TITCP013_CLM_VR_3</td>
<td>1</td>
<td>M</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>2</td>
<td>DSQ8ESQL</td>
<td>1</td>
<td>0</td>
<td>PROD</td>
<td></td>
<td>TITCP013_CLM_VR_3</td>
<td>1</td>
<td>M</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ACCESSCREATOR</th>
<th>ACCESSNAME</th>
<th>INDEXONLY</th>
<th>SORTN</th>
<th>SORTN</th>
<th>SORTN</th>
<th>SORTN</th>
<th>SORTC</th>
<th>SORTC</th>
<th>SORTC</th>
<th>SORTC</th>
<th>TSLOC/&lt;MODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBA</td>
<td>TITCP012_02</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>S</td>
</tr>
<tr>
<td>DBA</td>
<td>TITCP012_02</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>S</td>
</tr>
<tr>
<td>DBA</td>
<td>TITCP012_02</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>S</td>
</tr>
<tr>
<td>DBA</td>
<td>TITCP013_02</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>S</td>
</tr>
<tr>
<td>DBA</td>
<td>TITCP013_02</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>S</td>
</tr>
<tr>
<td>OBA</td>
<td>TITCP013_02</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>S</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TIMESTAMP</th>
<th>PREFETCH</th>
<th>EVAL</th>
<th>MIXOPSEQ</th>
<th>COLLID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1994031315265939</td>
<td>L</td>
<td>0</td>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>1994031315265939</td>
<td>L</td>
<td>1</td>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>1994031315265939</td>
<td>L</td>
<td>2</td>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>1994031315265939</td>
<td>L</td>
<td>3</td>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>1994031315265939</td>
<td>L</td>
<td>4</td>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>1994031315265939</td>
<td>L</td>
<td>5</td>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>1994031315265939</td>
<td>L</td>
<td>6</td>
<td>Q</td>
<td></td>
</tr>
<tr>
<td>1994031315265939</td>
<td>L</td>
<td>7</td>
<td>Q</td>
<td></td>
</tr>
</tbody>
</table>