

"Slug Hunting - - Using SAS to Evaluate Your Database Access Strategies in DB2"

Kathe Richards
Technical Environment Support Analyst
Texas State Comptroller's Office

Do you ever sit there waiting for your 25 rows of data from a DB2 table to return and wonder "What is it doing in there?" Here are some ways to find out and some suggestions about what to do next.

DB2 (and other relational DBMSs, too, sometimes) seems occasionally to pick up a reputation for being a "sluggish" performer. SAS occasionally gets this reputation, too, and in neither case is it necessarily deserved. As with any programming language, you can write really bad code if you work at it. Also, sometimes what you need to do is just not going to be all that easy to do. But with a little background about how things work and a little peek into the insides, you can often do things that improve even pretty good code.

How DB2 Works

DB2 is a relational database management system, which means that it adheres fairly closely to the rules set up by Dr. Codd and Dr. Date sometime in the 80's. Logically related groups of data are called databases. The data groups are stored as tables. The tables have rows and columns. Rows are unique. The columns containing the values that assure the rows are unique are called "primary keys". Information in a table is supposed to be somewhat logically consistent with the business purpose of the data in the table. Rows in tables can be "related" by matching specific column values with column values in rows in other tables in various. The columns that can be used to "relate" data in two or more tables are called "foreign keys".

DB2 uses physical structures called indexes to facilitate access to table data. This strategy is designed to speed up performance where possible. Using indexes, DB2 can find specific data pages more quickly and avoid having to read every row in a table. Indexes are typically built by the data base administrators (DBAs) on the basis of information provided to them by the developers of the applications that will be using the data. Among the most productive methods of improving performance of queries is to structure your code to use the indexes.

One of the features of DB2 (and other relational database management systems) is that to get the data out, you will have to use a language called SQL (for Structured Query

Language.) It's a limited set of syntax that is very flexible, very elegant, and occasionally very tricky. Its most striking characteristic is that it is non-procedural, meaning that you get to say what you want, but you don't get to say how to go about getting it. The DBMS engine decides this. Because of the rule that DBMSs use SQL, most development tools, including SAS, provide some interface so that SQL can be passed to the database.

In some cases, the development tool will attempt to relieve the user of the responsibility of actually writing the SQL, and SAS provides this, too, in the Data Manager Access Descriptor and View Descriptor feature. Often these interfaces take the form of "point and click" graphical layouts that generate the SQL resulting from your choices in the background.

Although these are handy, if you find you've constructed something that is not performing well, you will want to know what the actual SQL looks like. If you can cut and paste the SQL into a simple SAS job, you can find out a lot about what's going on in there. For this reason, it makes a good deal of sense in SAS to use PROC SQL and SQL pass through. Then you've got the code right there. Best advice: go ahead, bite the bullet, and learn SQL. It isn't that hard and it will look good on your resume.

Note that the syntax for PROC SQL and DB2's SQL is slightly different. They also operate on different data stores. PROC SQL operates on SAS datasets and DB2's SQL operates on DB2 tables. When you are using the pass through, you are connecting to DB2, passing a SQL statement of the DB2 variety, and getting back a result set in the form of a table which SAS puts in a SAS dataset. At this point you can operate on this SAS dataset in any way you would use any other SAS dataset, including using PROC SQL or using DATA step code.

How DB2 Decides What to Do

One of the big advantages of a DBMS is that somebody else is taking care of all the details like back-up and recovery, logging, reorganizing data structures, designing data stores, providing for security, and generally keeping things going. In DB2, much of the information you would possibly need to tell what is happening inside the DBMS will be stored in a special database called SYSIBM. This database contains tables that describe in great detail how DB2 is configured, how the data is stored, how much of it is where, who may update it, and any number of additional fun details.

The DB2 optimizer uses some of this information, along with analysis on the SQL code under consideration, to explore the possible access paths to the data and choose the one that is least expensive. You have access to the SQL code, of course, and interestingly, you also typically have access to the catalog tables in SYSIBM. General experience shared by others is often helpful additional knowledge to have. You will not, however, have exact knowledge of the algorithms the optimizer uses since that's proprietary

information (and besides, it changes from release to release.) But you can get DB2 to tell you a little about what the optimizer decided.

How You Can Find Out What DB2 Decided

Fortunately, there is within the syntax of DB2 a command called EXPLAIN. The official syntax is:

```
EXPLAIN [PLAN|ALL] {SET QUERYNO=integer}
      FOR explainable-sql-statement
```

An example of usage is:

```
EXPLAIN PLAN SET QUERYNO = 1000 FOR
  SELECT L.TP_ID, L.TAX_ID, L.LOC_NR, L.LOC_SQ, L.SIC_CD,
         L.PRMT_STA_CD,
         L.LOC_STOP_DT      AS OOBDATE,
         B.BUS_LOC_NM       AS LOC_NAME,
         B.CNTC_INFO_ID     AS CNTC_ID,
         L.LOC_FRST_BUS_DT  AS FRSTSLDT,
         L.LOC_PRMT_ISS_DT  AS PISSUEDT
      FROM TAX.VLOCATION01 L, TAXPYR.VBUS_LOC01 B
      WHERE L.TAX_ID = 26 AND
             L.PRMT_STA_CD = 'A' AND
             L.LOC_STOP_DT IS NULL AND
             L.BUS_LOC_ID = B.BUS_LOC_ID;
```

You could execute this through any interface that will accept SQL and pass it to DB2, such as QMF, various report generation tools and application environments, and, of course SAS.

In SAS, the syntax looks like this:

```
PROC DBLOAD DBMS=DB2;  SSID=DSNA;
```

```
  * This step cleans out any leftovers in your plan_table;
```

```
SQL
DELETE FROM KRIC459.PLAN_TABLE;
```

```
  * This step gets DB2 to execute an explain on the SQL
  * statement. Put the statement to be explained in the lines
  * following "FOR" and end with a semicolon;
  * The results of the explain will be written to your
  * plan_table.
  * The query number for it will be the number you assigned.
  *;
```

```

SQL
EXPLAIN PLAN SET QUERYNO = 1 FOR
SELECT
ITSA.VGE_STATE01.ST_CD ,
ITSA.VGE_STATE01.GE_ID
FROM
ITSA.VGE_STATE01
WHERE
( ITSA.VGE_STATE01.ST_CD = 'TX' );

```

Notice the reference to the object "plan_table" in the comments. This is where DB2 is going to put your information. If you do not have one, your job will end with an error. How do you get one? Well, you can ask your DBA to define one for you, or you can build your own.

In our environment, the DBAs have coded a QMF procedure in each subsystem that individuals can run to create their plan tables. This is a very good arrangement, since they have ready access to the syntax that will be needed, they are in a good position to update the syntax if it changes in a new release, and they will almost certainly have appropriate authority to do it whereas you may not. If you really want to do it yourself, the command you will need is CREATE TABLE and you will have to specify each column in the table and its characteristics. It will be prefixed by your identifier as the database name.

What You Will Get

The result of the EXPLAIN command is an update to your plan table. This will not be very interesting to you until you look at it, however, so I add the following steps to my SAS code to have a look at the plan table:

- * This step reads the contents of your Plan_table and puts it
- * in a SAS dataset named EXPLAIN. We then sort it for easier
- * reading and print it. ;

```

PROC SQL;
  CONNECT TO DB2 (SSID=DSNA);
  CREATE TABLE EXPLAIN AS
    SELECT * FROM CONNECTION TO DB2
      (SELECT * FROM KRIC459.PLAN_TABLE WHERE QUERYNO < 20);
  %PUT &SQLXMSG;
  %PUT &SQLXRC;

PROC SORT;  BY QUERYNO QBLOCKNO PLANNO;

```

```

PROC PRINT DATA=EXPLAIN(DROP=IBM_SERV REMARKS APPLNAME
PROGNAME) NOOBS;
  BY QUERYNO; ID QUERYNO;

TITLE1 'OUTPUT FROM EXPLAIN ON GERI 4';
TITLE2 'DB2 SYSTEM -- KER JOB';
TITLE4 &SYSDATE;

```

The output is going to look something like this:

```

OUTPUT FROM EXPLAIN ON GERI 4
      DB2 SYSTEM -- KER JOB

      17AUG98

      Q
      Q B          C          T          A M A A I S S S S S S S S T T
      U L P M      R          T E C C C D R R R R R R R R R L M
      E O L E      E          A S H S S X N N N N C C C C C S
      R C A T      A          B S C S S O _ _ _ _ _ _ _ K T
      Y K N H      T          N T O C N N U J O G U J O G M A
      N N N O      O          O Y L R A L N O R R N O R R O M
      O O O D      R          E

      1 1 1 0 ENTRPRSE GOV_ENTY 1 R 0      N N N N N N N N N N IS
1998081714464

```

<< partial listing >>

SAS has, of course, cleverly condensed the column names to 8 bytes since that is his preference. Never mind that it might not be yours and certainly isn't DB2's. We can code around this in various ways, of course, but I will leave that detail to you. If you want to know the actual column names, you can look them up in one of IBM's handsome reference works such as the "Application Programming and SQL Guide" or you could find a DBA who has one of the wonderful charts or references from one of the tool vendors that shows all the system tables, or you could query the system tables yourself.

Meanwhile, let's have a look at some of the information DB2 is giving us here. The variable QUERYNO is the number that you (optionally) give it in the EXPLAIN PLAN statement. If you already have output from an explain with that query number, your results for this explain will be mixed with the results from the previous one and the only way you will be able to tell them apart is by looking at the TIMESTAM (timestamp) column. This is messy, so it helps to take some care to keep your plan table cleaned up.

For each query, you will have a row of output for each of the tables named in the query and a row for any ORDER BY, GROUP BY or any intermediate tables DB2 has to build

for you to complete your request. If you have a subquery, the column QBLOCKNO will contain tell you which lines refer to the main query and which lines refer to the subquery. The PLANNO column will identify the order in which the steps were executed within the QBLOCKNO (the table join order for that query or subquery).

The METHOD column will tell you what join method or sort is used for this step. The value 0 (zero) indicates that this is the first (or outer) table accessed, continuation of previous table accessed, or not used. The value 1 indicates a nested loop join. For each qualifying row in the outer table, all matching rows in the inner table are found and applicable WHERE clause conditions applied. A 2 here indicates a merge scan join, a 3 indicates additional sorts are needed to take care of ORDER BY, GROUP BY, SELECT DISTINCT, or UNION or some other requirement of the code. This step does not access a new table. A 4 indicates a hybrid join. Extensive discussions of these items can be found in the SQL Guide and most other references but generally, you will be able to find out what you need to know by looking at other information in the plan table output for this query.

The additional columns of the output that are going to be most interesting to us are: TNAME, TABNO, ACCESSTY, MATCHCOL, ACCESSNA, and INDEXONL. (In our environment, CREATOR and ACCESSCR are also helpful since by convention our DBAs use these fields to indicate the database in which the tables and indexes are found. This may not be the case for your installation.

TNAME is the name of the table being read, TABNO is the order in which it appears in your SQL statement (handy if you have a subquery that accesses the same table as the main query), ACCESSTY is the method used to access the new table, MATCHCOL is the number of index key columns used, ACCESSNA is the name of the index used, and INDEXONL indicates if only index entries are accessed for a step.

Of these columns, the ones you want to pay special attention to are ACCESSTY and MATCHCOL. ACCESSTY can have 9 values. What you prefer to see in this column is an I or an I1 indicating an index is being used for access to this table. What you prefer not to see is an R, which indicates that a table space scan is occurring. In other words, in order to get what you want, DB2 is having to read each row of the table. Even if you have an I or I1 (or another index-based access type) it may not be the best you can do. Look next at the MATCHCOL. If there is a zero in this column, you will be doing an index space scan, which may be pretty resource consuming, too, depending upon how big the table is and which index you're looking at.

Here's a snippet of output from an explain on a pretty complex SQL statement:

Q						A	M	A	A	I
Q	B			C		C	A	C	C	N
U	L	P	M	R		C	T	C	C	D
E	O	L	E	E	T	T	E	C	E	E
R	C	A	T	A	N	A	S	H	S	X
Y	K	N	H	T	A	B	S	C	S	O
N	N	N	O	O	M	N	T	O	C	N
O	O	O	D	R	E	O	Y	L	R	L

1	1	1	0	WORKMGR	WORK_POOL_EMP	3	I	2	WORKMGR	WRKPOLE3	Y
	1	2	1	WORKMGR	WORK_POOL_ASGN	2	R	0			N
	1	3	1	WORKMGR	OFFC_ASGN	1	I	1	WORKMGR	OFFASGN1	N
	1	4	1	TAXPYR	TAXPAYER	4	I	1	TAXPYR	TAXPAY1	N
	1	5	1	TAXPYR	TP_NAME	5	I	1	TAXPYR	TPNAME1	N

This one is telling us that the first table accessed is WORK_POOL_EMP, an index is being used, and two columns of the index are being used. The name of the index is WRKPOLE3 and all of the data requested from this table can be read from the index so no data rows are read.

The second table accessed is WORK_POOL_ASGN, and we are not as happy with this outcome. It is doing a table space scan. If this is a large table, this could be trouble for us. The last three tables are using indexes with matching columns, so we can expect that these are performing pretty well.

How You Can Find Out What Else You Need to Know

At this point it would clearly be helpful to know a little more about the tables involved here. We would like to know how big these tables are, what the columns in the tables are, what other indexes might be available, and what additional information we might be able to provide to get a little better strategy. The output in the plan table gives us the information that we need to query the SYSIBM tables and find out a lot of this information. Here is some additional SAS code that will do this for you.

This step will give you descriptive information about the tables accessed:

```
PROC SQL;
  CONNECT TO DB2 (SSID=DSNA); *<----USE DSNT FOR TEST
  CREATE TABLE COLUMNS AS
  SELECT * FROM CONNECTION TO DB2
    (SELECT * FROM SYSIBM.SYSCOLUMNS WHERE TBCREATOR
      IN
        (SELECT CREATOR FROM KRIC459.PLAN_TABLE
          WHERE QUERYNO = 1)
        AND TBNAME IN
          (SELECT TNAME FROM KRIC459.PLAN_TABLE
            WHERE QUERYNO = 1)
      ORDER BY TBNAME, COLNO );
  %PUT &SQLXMSG;
  %PUT &SQLXRC;
```

And this step will give you information about the indexes on these tables:

```
CREATE TABLE SYSTABS AS
  SELECT * FROM CONNECTION TO DB2
  (SELECT A.CREATOR, A.TBNAME, A.NAME, B.COLNAME, B.COLSEQ,
    B.ORDERING, A.UNIQUERULE, A.CLUSTERING
    FROM SYSIBM.SYSINDEXES A, SYSIBM.SYSKEYS B
    WHERE A.TBNAME IN
    (SELECT TBNAME FROM KRIC459.PLAN_TABLE
     WHERE QUERYNO = 1)
    AND A.CREATOR IN
    (SELECT CREATOR FROM KRIC459.PLAN_TABLE
     WHERE QUERYNO = 1)
    AND A.NAME = B.IXNAME
    AND A.CREATOR = B.IXCREATOR
    ORDER BY 1, 2, 3, 5);

%PUT &SQLXMSG;
%PUT &SQLXRC;
DISCONNECT FROM DB2;
QUIT;
```

And here's a tiny bit of code to format and print the results:

```
PROC PRINT DATA=COLUMNS (DROP= HIGH2KEY LOW2KEY); BY
TBNAME; ID TBNAME;
PROC PRINT DATA=SYSTABS; BY CREATOR TBNAME; ID TBNAME;
//
```

Here are some snippets from the output from these queries when applied to the query with the table space scan above. (I have not included all the output but can make it available along with the complete SAS code for the assembled job to anyone who is interested.)

From the SYSTABLES query:

			C				C
			O				O
			L				L
			E	S	N		L
			N	C	U		C
			G	A	L		A
			T	L	L		R
			H	E	S		D
WORK_POOL_ASGN	OFFC_ASGN_ID	1	DECIMAL	13	0	N	2729
	WRK_POOL_ID	2	DECIMAL	13	0	N	78
	SETU_DT	3	DATE	4	0	N	248
	TSK_TY_CD	4	CHAR	8	0	Y	30

WORK_POOL_EMP	USER_ID	1	CHAR	8	0	N	849
	WRK_POOL_ID	2	DECIMAL	13	0	N	220
	END_DT	3	DATE	4	0	Y	109
	BEG_DT	4	DATE	4	0	N	198
	POSN_CD	5	CHAR	8	0	N	1

This partial output shows some of the information available for each table. The COLCARD (column cardinality) column is of special interest because, along with the index information, it will give us an idea of how big the tables are.

TBNAME	NAME	COLNAME	COLSEQ	ORDERING	UNIQU
WORK_POOL_ASGN	WKPOLAS1	OFFC_ASGN_ID	1	A	P
WORK_POOL_ASGN	WKPOLAS1	WRK_POOL_ID	2	A	P

TBNAME	NAME	COLNAME	COLSEQ	ORDERING	UNIQUERU
WORK_POOL_EMP	WRKPOLE1	USER_ID	1	A	P
WORK_POOL_EMP	WRKPOLE1	WRK_POOL_ID	2	A	P
WORK_POOL_EMP	WRKPOLE1	BEG_DT	3	A	P
WORK_POOL_EMP	WRKPOLE2	END_DT	1	A	U
WORK_POOL_EMP	WRKPOLE2	WRK_POOL_ID	2	A	U
WORK_POOL_EMP	WRKPOLE2	USER_ID	3	A	U
WORK_POOL_EMP	WRKPOLE2	BEG_DT	4	A	U
WORK_POOL_EMP	WRKPOLE3	USER_ID	1	A	D
WORK_POOL_EMP	WRKPOLE3	END_DT	2	A	D
WORK_POOL_EMP	WRKPOLE3	WRK_POOL_ID	3	A	D

This partial output from the same two tables indicates which indexes are available on each table and describes them. An additional column available from this output is CLUSTERING which may have the values of Y and N and indicates, when the value is Y, that the table will tend to be in approximately this order, so DB2 will tend to prefer this access when possible. We also have quite a bit of information about the indexes here. On the first table we have only 1 index, WKPOLAS1 consisting of two columns and they make up the primary key of the table. The second table has three indexes, WRKPOLE1, WRKPOLE2 and WRKPOLE3, each of which is made up of several columns. WRKPOLE1 is the primary key, WRKPOLE2 can be expected to be unique, and WRKPOLE3 may have duplicate values.

Looking at this information in conjunction with the output from the explain, we might presume that the query in question is not supplying information about OFFC_ASGN_ID, the first column in the only index available on this table. This is probably what is slowing the query down.

What to Do Now – Things That Might Help

In this particular case, of course, you would want to try to find some way to either supply the information needed by the index or restructure the query to use the index you have. Failing that, you could look for some other column that might be a likely candidate for a supporting index. Then you would need to persuade the DBA in charge of this table that this would be a good idea.

In general, the information you find out from explains can be used in pretty much the same way. Now that you know what DB2 thinks he's going to do when he sees your query, you can apply all you know about good SQL coding practices to the project of changing his mind. If you don't feel that confident about your store of information about good SQL coding practices (an interesting topic for study in itself), the output from this SAS job along with a listing of the query itself will be handy to bring with you when you go ask somebody to help you.

Things that are likely to help are supplying data in the query that will make the optimizer more likely to choose a favorable index, restructuring the logic of the SQL, and persuading the DBAs to change the structure of the data to improve your access path.

As SAS users, though, we have additional power at our disposal. We can consider the possibility of restructuring the task so that SAS is doing some of the work outside of DB2. We have been very successful in overcoming some of DB2's quirks by the creative application of SAS strategies (but that's the subject of another paper.)

SAS is a registered trademark of the SAS Institute Inc., Cary, NC, USA.

Other brand and product names are the trademarks of their respective companies.

For a complete listing in electronic format of the SAS code in this paper, contact me. My e-mail address is Kathe.Richards@cpa.state.tx.us.