

SAS/ACCESS TECHNIQUES FOR LARGE DB2 DATABASES

Craig James Gavin Jr.
Atlantic Research Corporation, Rockville Maryland

ABSTRACT

The SAS/ACCESS interface software to Relational Database Management Systems provides the data storage integrity of a RDBMS and the flexibility of the SAS system's data manipulation and reporting attributes. When working with large databases performance is a key factor. Some extensive queries could compromise the usefulness of the SAS system. When coding with SAS/ACCESS using a RDBMS such as DB2, the applications programmer must look beyond the basic techniques for inserting, updating and extracting data from tables. This paper will present ways to overcome complex problems that users of SAS/ACCESS interface software may encounter. It also will address recent SAS/ACCESS software tools available through SAS Release 6.07.

INTRODUCTION

Thanks to the latest release of SAS 6.07, the applications programmer is supplied with user-friendly tools to access large DB2 tables. The focus of this paper is to introduce techniques that deal with these tables. Although the RDBMS used in this paper is DB2, the same theories will apply to any RDBMS SAS/ACCESS interfaces.

Some of the topics this paper will discuss include:

- Automating the creation of ACCESS descriptors and views.
- Fetching DB2 data using a SAS dataset for subsetting.
- Inserting SAS data into DB2 tables.
- DB2/SAS performance issues

The following are assumptions the author takes for granted:

1. The reader has a general knowledge of Base SAS and SAS/ACCESS software.

2. The user has DB2 privileges to create and maintain DB2 tables.
3. Since DB2 is an IBM mainframe product, the program examples are written to execute on the MVS operating system.

AUTOMATING THE CREATION OF SAS/ACCESS DESCRIPTORS AND VIEWS

One of the greatest improvements in SAS's Release 6.07 is the ability to create ACCESS descriptors in batch mode. With the addition of PROC SQL's Pass-Through Facility the user now has the tools to automate the creation of ACCESS descriptors and views.

Creating Descriptors and Views

* LIBNAME is the SAS data library that stores the ACCESS descriptors and views ;

```
LIBNAME ACCLIB '..SAS library..' DISP=OLD ;
```

* Step 1

Get table names from the DB2 System Catalog ;

```
PROC SQL ;
  CONNECT TO DB2 (SSID=DB2) ;
  CREATE TABLE T_NAMES AS
  SELECT * FROM CONNECTION TO DB2
    (SELECT NAME
     FROM SYSIBM.SYSTABLES ) ;
  %PUT &SQLXMSG ;
  DISCONNECT FROM DB2 ;
QUIT ;
```

```
FILENAME TEMP '&TEMP' RECFM=FB LRECL=80
BLKSIZE=6160 ;
```

* Step 2

Create PROC DATASETS statements to delete the descriptors that already exist in the ACCLIB library ;

```
DATA _NULL_ ;
  SET T_NAMES END=LAST ;
```

```
FILE TEMP ;
IF _N_ = 1 THEN DO ;
  PUT 'PROC DATASETS LIBRARY=ACCLIB' /
    'MEMTYPE=(VIEW ACCESS) NOLIST ;'
  / ' DELETE ' ;
END ;
PUT NAME @ ;
IF LAST THEN PUT ; RUN ; ;
RUN ;
```

*** Step 3**

Create PROC ACCESS statements to rebuild or build Access descriptors and views for each table in the database ;

```
DATA _NULL_ ;
  SET T_NAMES END=LAST ;
  FILE TEMP MOD ;
  IF _N_ = 1 THEN DO ;
    PUT / 'PROC ACCESS DBMS=DB2;' ;
  END ;
  PUT 'CREATE ACCLIB.' NAME +(-1)'.ACCESS;' /
    ' TABLE = ' NAME +(-1) ';' /
    ' CREATE ACCLIB.' NAME +(-1)'.VIEW;' /
    ' SELECT ALL;' ;
  IF LAST THEN PUT 'RUN;' ;
RUN ;
```

*** Step 4**

Include the PROC DATASETS and PROC ACCESS statements ;

```
%INCLUDE TEMP ;
```

DB2 maintains a set of tables known as the system catalog. The catalog describes tablespaces, tables, columns, indexes, privileges, and application plans information. Data in the catalog are available to all authorized users of the database. The catalog tables are updated by DB2 in response to SQL data definition statements, GRANT and REVOKE statements, and other specific commands. This provides information on the users database as it exists. In Step 1, all DB2 table and view names are fetched from the catalog table SYSIBM.SYSTABLES. Some DB2 installations may not grant SELECT privileges directly to the catalog. If this is the case, DB2 views are usually created to allow access. The Database Administrator should be able to identify any view name of the catalog.

In Step 2, all descriptors and views that exist in the

ACCLIB library will be deleted if a table or view currently exists in the catalog. (Since these descriptors and views will be recreated, deleting them should not cause a problem.)

PROC DATASETS statements are created and stored in the temporary file "TEMP" for later execution. By performing this step a DB2 table or view can be altered or recreated, and the descriptor will be redefined to reflect the change. If a DB2 table or view is dropped from the database, this program will not drop an existing descriptor.

In Step 3, the Access descriptors and views are created using the PROC ACCESS procedure. These statements are generated and appended to the temporary file "TEMP" which contains the PROC DATASETS commands.

Step 4 executes the commands stored in the temporary file "TEMP". This code will create descriptors and views for every DB2 table and view that exists in the user's database. Every column will be included for each descriptor and view.

Some installations may design their SAS/ACCESS software to limit access to certain database information. If data security is properly maintained at the DB2 level, passwords and limiting columns are usually not necessary.

PROBLEMS WITH SAS/ACCESS WHEN USING LARGER DB2 TABLES

The theory behind SAS/ACCESS is to make the connection between SAS and DB2 transparent. Once a view descriptor is created for a DB2 table or view, it appears to the SAS user as a normal SAS data view. A descriptor view is not a SAS dataset but a vehicle to DB2. The performance of the program utilizing the view, will not only depend upon the SAS coding, but also the method in which the DB2 optimizer fetches the DB2 data.

Every installation has its own performance demands, whether it's CPU time or real time. When accessing larger tables, the SET statement without a WHERE clause on an indexed field will surely maximize performance limits. Another way of pushing performance limits is by using a WHERE clause on a non-indexed fields. The DB2 optimizer must do a tablespace scan to fetch the information

SAS requests.

DB2 Table A	
Sample	Date
AAA	11/91
BBB	01/92
CCC	05/92
DDD	12/91
EEE	06/90

Figure 1

so. Suppose there is a SAS dataset (Figure 2) with data the user wishes to match with the DB2 table A.

SAS Dataset SAMPLE	
Sample	
BBB	
XXX	
YYY	
AAA	

Figure 2

```
DATA TESTINDX ;
  SET ACCLIB.A ;
  WHERE DATE = '12/91' ;
```

A tablespace scan reads every record in the tablespace to satisfy the request. This is a costly and inefficient method of fetching DB2 data.

An even worse method is to SET or MERGE with an IF statement for subsetting.

```
DATA TESTINDX ;
  SET ACCLIB.A ;
  IF SAMPLE = 'DDD' ;
```

SAS will request from DB2 every record in the table. SAS will then SET each record into the program data vector and apply the IF statement.

The optimal method of extracting DB2 data is by using the WHERE clause on a DB2 indexed value.

```
DATA TESTIDNX ;
  SET ACCLIB.A ;
  WHERE SAMPLE = 'DDD' ;
```

The WHERE clause is passed to the DB2 optimizer and the optimizer utilizes the index. A search by an index is considerably faster than a tablespace scan. The object of the SAS code is to utilize DB2 indexes whenever possible.

TECHNIQUES FOR EXTRACTING DATA FROM LARGE DB2 TABLES BY INDEXES

Due to the variety of ways in fetching DB2 table data, there is no single efficient method for doing

Given that the best method of querying DB2 data is by using a WHERE clause on an indexed field, storing the SAMPLE values into SAS macro variables can be used to build a WHERE statement:

```
%MACRO WRITE ;
%LET VAR =WHERE ;
%DO I = 1 %TO &NUMWHERE ;
  &&&VAR&I
%END ;
%MEND WRITE ;

DATA _NULL_ ;
  LENGTH Y $20 ;
  SET WORK.SAMPLE NOBS=TOTAL
  END=DONE ;
  IF _N_ = 1 THEN CALL
    SYMPUT('NUMWHERE',PUT(TOTAL,2.));
  IF NOT DONE THEN
    Y = ""||SAMPLE||"," ;
  ELSE Y = ""||SAMPLE||"" ;
  CALL SYMPUT('WHERE'||LEFT(_N_),Y);
RUN ;
```

```
DATA SAMPLES ;
  SET ACCLIB.A ;
  WHERE SAMPLE IN (%WRITE) ;
RUN ;
```

Executing the above code will produce the following:

```
DATA SAMPLES ;
  SET ACCLIB.A ;
  WHERE SAMPLE IN ('BBB','XXX','YYY','AAA')
```

```
;
RUN;
```

This is an easy application if the user's look-up data is a one to one match with the DB2 table data. Since most DB2 tables are linked by some form of relationship, fetching may be more complex. In the next example there are two tables (Figure 3). The table named SAMP contains blood test results for a patient. The blood is given a code when it is drawn from the patient. That field is called SAMPL_ID. In order to link the SAMPL_ID to patient information, the table includes a patient identifier (SUBJ_ID) code. There are two indexes for this table. The primary index is SAMPL_ID and the secondary index is the SUBJ_ID identifier. The SUBJECT table contains information about the patient. There is only one record for each patient (SUBJ_ID). A unique index exists for SUBJ_ID.

SAMP	SUBJECT
SAMPL_ID	SUBJ_ID
SUBJ_ID	RACE
TEST_DT	NAME
RESULT	BIRTH_DT

Figure 3

Suppose the user was interested in patient information for the sample AAA. To fulfill this request, a JOIN between the result of the SAMP table and the matching SUBJ_ID in the SUBJECT table will produce the patient information. Here are some examples of the "do's and don'ts":

```
DATA SUBJINFO ;
MERGE ACCLIB.SAMP
      ACCLIB.SUBJ ;
BY SUBJ_ID ;
WHERE SAMPL_ID = 'AAA' ;
```

This example is a don't, SAS will send the WHERE clause to the DB2 optimizer for both tables. Since SUBJECT doesn't have a SAMPL_ID field the program would return with an error.

Another method is to use PROC SQL.

```
PROC SQL ;
SELECT *
FROM ACCLIB.SAMP A,
      ACCLIB.SUBJECT B
WHERE A.SAMPL_ID = 'AAA' AND
      A.SUBJ_ID = B.SUBJ_ID ;
```

This code will fetch the proper results, but will do all of the join work in SAS. SAS will parse the WHERE clause and request two sets of data from DB2:

1. The column-to-character string portion (WHERE A.SAMPL_ID = 'AAA') for subsetting.
2. Every record from the SUBJECT table.

The second step, requesting every record from the SUBJECT table, will be a time consuming task. This is done because SAS cannot pass a column-to-column join to DB2. The result of the SAMP query and every record in the SUBJECT table, will be joined at the SAS level. To investigate how queries parse, use the \$DB2DEBUG system option. This option will print exactly what was passed to DB2 in the SAS log.

The optimal solution is to have DB2 utilize the indexes while doing the join. This would mean getting the entire WHERE clause to the DB2 optimizer. In SAS Release 6.06 this was impossible. In Release 6.07, PROC SQL includes the Pass Through facility which has this capability. The Pass Through Facility sends the entire SQL query between the brackets to DB2.

```
PROC SQL ;
CONNECT TO DB2(SSID=DB2) ;
CREATE TABLE SUBJINFO AS
SELECT * FROM CONNECTION TO DB2
(SELECT *
FROM SAMP A,
      SUBJECT B
WHERE A.SAMPL_ID = 'AAA' AND
      A.SUBJ_ID = B.SUBJ_ID) ;
%PUT &SQLMSG ;
DISCONNECT FROM DB2 ;
QUIT ;
```

The benefit of this is that DB2 will do all of the join processing. Once DB2 completes its work, the data are then stored into the SAS work file SUBJINFO. This SAS file is like any other SAS

dataset.

The relational queries were accomplished by requesting a character string to equal the variable `SAMPL_ID`. Suppose the user had to match `SAMPL_ID` with a value in a SAS dataset. This would mean a three-way join between the SAS dataset, `SAMP` table, and `SUBJECT` table. In this example the SAS dataset named `MATCHME` contains `SAMPL_ID` codes:

```
DATA MATCHME ;
  INPUT @1 SAMPL_ID $CHAR3. ;
CARDS ;
CCC
XRK
;

PROC SQL ;
  CONNECT TO DB2 ;
  SELECT *
    FROM CONNECTION TO DB2
      (SELECT A.SAMPL_ID, B.STUDY_ID,
        B.SUBJ_ID
      FROM SAMP A,
        SUBJECT B
      WHERE A.SAMPL_ID = B.SAMPL_ID )
      AS DB2TMP,
  WORK.MATCHME AS SASTMP
  WHERE SASTMP.SAMPL_ID =
    DB2TMP.SAMPL_ID;
  %PUT &SQLXMSG ;
  DISCONNECT FROM DB2 ;
QUIT ;
```

This program will do two joins before getting the final result. The first join is done at the DB2 level, all `SAMPL_ID`s in the `SAMP` table are joined with all `SAMPL_ID`s in the `SUBJECT` table. Once the DB2 join is completed, the data is stored in a SAS work dataset. The second join will match the data in the work dataset (from the DB2 join) to the created `MATCHME` SAS dataset.

The first join of this query is very costly. Since it is impossible to pass the subsetting SAS dataset `MATCHME` to the DB2 optimizer. The large DB2 tables, `SAMP` and `SUBJECT`, are joined by DB2. SAS will then fetch the results of the joined tables and do a second join with the `MATCHME` dataset. A solution to this problem is to perform a similar technique as in the first example of this section.

By creating the `WHERE` statements with SAS macro variables, the column-to-character string portion, along with the joining of the `SAMP` and `SUBJECT` tables, would be passed to DB2. Creating these macro variables would not be complex.

```
PROC SQL ;
  CONNECT TO DB2 ;
  SELECT * FROM CONNECTION TO DB2
    (SELECT *
     FROM SAMP A,
       SUBJECT B
     WHERE A.SAMPL_ID IN ( %WRITE ) AND
       A.SAMPL_ID = B.SAMPL_ID ) ;
  %PUT &SQLXMSG ;
  DISCONNECT FROM DB2 ;
QUIT ;
```

The `%WRITE` macro would fill in the `SAMPL_ID` values in the SAS dataset.

Writing out the `WHERE` clause is not always feasible. For example there are two values in the SAS dataset. The user needs all the data in the `SAMP` table that is between those two values.

```
WHERE SAMPL_ID BETWEEN 'AAA' AND 'DDD'
OR
  SAMPL_ID BETWEEN 'MMM' AND 'PPP'
```

The macro language could again be used to build the `WHERE` clause. If this is done there are certain performance factors when passing such a statement to DB2. In DB2 release 2.2 there is a limit of 99 `BETWEEN` statements in using the indexes. The DB2 optimizer will parse the `SELECT` statement to calculate the optimal method for performing the request. If the number of `BETWEEN` statements is over 99, a tablespace scan will be performed on that table and the index will be ignored.

One solution would be to create an intermediate look-up DB2 table that can be used to join tables. This would ensure all querying requests are done at the DB2 level.

Suppose there is a SAS dataset named `TMPTABLE` (Figure 4) in the work data library. This dataset has five variables called `JOBID`, `SAMPST`, `SAMPEND`, `SUBJST` and `SUBJEND`. This dataset

could have been created by a user of an interactive application or by another application.

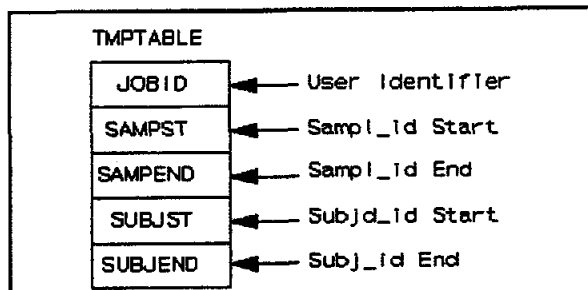


Figure 4

By inserting the contents of TMPTABLE into one of these intermediate DB2 tables (Figure 5) an efficient join can be performed.

The DB2 table SAMPLIST will be created to specify ranges or matches with the SAS dataset TMPTABLE. The SUBJLIST table will be created for fetching a range or match of TMPTABLE's SUBJ_IDs. The JOBID variable can act as a reference code that the program would use to identify the proper look-up records. This will be necessary if the temporary tables has multiple users. The JOBID code could be created dynamically using the macro language's system macros.

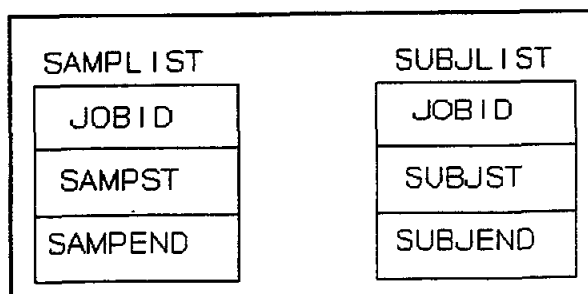


Figure 5

Once the SAS data is in the temporary DB2 table it can be Joined with the larger tables.

```
* To query by SAMPL_ID ;
%LET JOBID=%SUBSTR(&SYSJOBID,1,3)
        &SYSDATE%SUBSTR(&SYSTIME,1,2)
```

```
%SUBSTR(&SYSTIME,4,2) ;
```

```
DATA TMPTABLE ;
  LENGTH JOBID $14. ;
  JOBID = SYMGET('JOBID') ;
  INPUT @1 SAMPST $CHAR3.
        @4 SAMPEND $CHAR3.;
  CARDS ;
AAACCC
TTTXXX
;
PROC APPEND BASE=ACCLIB.SAMPLIST ;
  DATA=TMPTABLE(KEEP=JOBID SAMPST
    SAMPEND) ;
```

```
* To query by SUBJ_ID ;
%LET JOBID=%SUBSTR(&SYSJOBID,1,3)
        &SYSDATE%SUBSTR(&SYSTIME,1,2)
        %SUBSTR(&SYSTIME,4,2) ;
```

```
DATA TMPTABLE ;
  LENGTH JOBID $14. ;
  JOBID = SYMGET('JOBID') ;
  INPUT @1 SUBJST $CHAR7.
        @9 SUBJEND $CHAR7.;
  CARDS ;
CA00001 CA01000
CB50000 CC10000
;
PROC APPEND BASE=ACCLIB.SUBJLIST ;
  DATA=TMPTABLE(KEEP=JOBID SUBJST
    SUBJEND) ;
```

Using PROC APPEND is not the only method of inserting data into a DB2 table. The PROC SQL Pass Through Facility has a means of executing INSERT statements for each record. If there is a considerable amount of data to be inserted, multiple executes becomes sluggish.

The data stored in the intermediate DB2 tables (SAMPLIST & SUBJLIST) can be deleted after each program execution. It may be prefer to leave the data in these tables to monitor the types of queries users are executing. The JOBID field will distinguish queries between users.

If the intermediate tables are accumulating queries, an index on JOBID will improve performance. The IBM DB2 utility program RUNSTATS (refer to IBM COMMAND and UTILITY Reference) should

occasionally be run on these tables due to their volatility.

Once the SAS data is in the intermediate DB2 table, joins between it and the database can be performed.

*If SAMPL_ID information ;

```
PROC SQL ;
CONNECT TO DB2 ;
SELECT * FROM CONNECTION TO DB2
(SELECT *
FROM SAMP A,
SUBJECT B,
SAMPLIST C
WHERE C.JOBID = %STR('%')&JOBID%STR('%')
AND A.SAMPL_ID BETWEEN
C.SAMPST AND C.SAMPEND AND
A.SUBJ_ID = B.SUBJ_ID ) ;
%PUT &SQLXMSG ;
DISCONNECT FROM DB2 ;
QUIT ;
```

* For SUBJ_ID information ;

```
PROC SQL ;
CONNECT TO DB2 ;
SELECT * FROM CONNECTION TO DB2
(SELECT *
FROM SAMP A,
SUBJECT B,
SUBJLIST C
WHERE C.JOBID = %STR('%')&JOBID%STR('%')
AND B.SUBJ_ID BETWEEN
C.SUBJST AND C.SUBJEND
AND B.SAMPL_ID = A.SAMPL_ID ) ;
&PUT &SQLXMSG ;
DISCONNECT FROM DB2 ;
QUIT ;
```

CONCLUSION

The objective of the paper is to offer solutions and general knowledge of SAS/ACCESS software when accessing larger RDBMS tables. If the programs in this paper cannot be applied to the reader's application, it is hoped that they will act as a midwife to their solutions. If any reader wishes to discuss this paper or any other SAS/ACCESS issues the author may be contacted at:

Craig J. Gavin Jr.
Atlantic Research Corp.
1301 Piccard Dr.
Rockville MD, 20850

ACKNOWLEDGEMENTS

The author would like to thank Phil Virgo for his focus and technical prowess.

SAS and SAS/ACCESS are registered trademarks of SAS Institute, Inc.

DB2 and MVS are registered trademarks of International Business Machines Corporation.

REFERENCES

1. SAS Technical Report P-221
SAS/Access Software: Changes and Enhancements
2. SAS Technical Report P-222
Changes and Enhancements to Base SAS Software
3. Gabrielle Wiorkowski, David Kull, DB2 Design & Development Guide, (Addison-Wesley Publishing 1990)
4. IBM DATABASE 2 Version 2, Command and Utility Reference, (IBM Corporation September 1988)