# SAS Connections to DB2: Tools and Techniques

**Judy Loren, Independent Consultant**

## Abstract

In the beginning there were Access Descriptors that allowed you to treat DB2 tables as if they were SAS datasets. Access Descriptors are simple to use, but take prohibitively long to run on large tables. Then came Pass-Through, so you could pass an entire SQL query directly to DB2 to execute. This allows DB2 to take advantage of its indexes, improving run time when all the information is in DB2. But what if you need to join a list of keys contained in a SAS dataset with a DB2 table (or tables)? This paper presents some suggestions based on real world experience using SAS as an access language for large DB2 databases.

## Acknowledgements

Right up front I want to give credit to the people with whom I have had the pleasure of working to come up with the techniques described in this paper.

Alan Dickson
Tom Finn
Craig Gavin
Paul Kent
Stephen Scott

## Introduction

To get SAS and DB2 to work together effectively, you need an array of tools and an understanding of when to use each of them. First I will discuss several techniques; then I will give you guidelines to help decide when to use them.

In the following discussion, I'm assuming you have some familiarity with SQL, and that you have a basic understanding of the following: DB2 tables, DB2 indexes, DB2 subsystems, SAS datasets, SAS indexes, flat files.

This paper is written about DB2 under MVS.

Most of the challenges involved in using SAS with DB2 are performance related, primarily CPU consumption and clock time. Inefficiencies in these two areas can be insignificant with tables containing only hundreds of records. But when you try to use access descriptors on tables with millions of records--well, if you ever did, it's probably still running.

By way of introducing the topic, here are some comparisons of CPU times for three ways of accomplishing the same merge. This is just to give you an idea how much it matters which tool you select.

| Tool | CPU |
|---|---|
| ACCESS DESCRIPTOR | 197 |
| DB2 JOIN | 25 |
| SAS MERGE | 7 |

### Access Descriptors

The first thing I should mention in discussing SAS and DB2 is access descriptors. These are like views that allow you to treat a DB2 table as if it were a SAS file. Now that I've mentioned them, forget them. When using large DB2 tables, the first thing you learn is DON'T USE ACCESS DESCRIPTORS. One reason is that when you use access descriptors, SAS doesn't deliver enough information to DB2 to allow it to make use of its indexes. As a result, everything is done as a table scan. Also, it seems that with Access Descriptors, there is some kind of record by record handshaking between SAS and DB2.

### DB2 Indexes

Indexes are a cornerstone of DB2 performance. An appropriate index can make a query run like greased lightning. For example, if you pass DB2 a hard coded value of a keyed field, DB2 will return the desired record(s) before you have a chance to ask. Without appropriate indexes, most queries will either not run at all because you exceed DB2 resource limitations, or will run until the system IPL.

The idea we have focused on when dealing with large tables in DB2 is to provide DB2 with the cues it needs to make it use its indexes. Most of the tools in this paper depend on the DB2 tables being indexed on the key field you are interested in. If that overhead hasn't been done for you, I recommend dumping the table to SAS and sorting there. This paper contains a technique for quickly extracting an entire table or portions of a table.

As an aside, I will mention indexes in SAS because the table you dump may be too big to sort. Indexing can work to your advantage in this case, but use it cautiously. Don't do a huge volume of direct access reads using SAS

indexes without reading David Sommer's paper from SUGI 20 entitled Efficient Techniques for Random Access to SAS Datasets. But SAS Indexes are another topic for another day. Back to DB2.

**Pass-Through Access**

If all the data you need are in DB2 tables and you just want to use SAS to manipulate it, you have a friend in Pass Through access. Pass Through is a feature of SAS PROC SQL introduced in 6.07 that allows you to construct a DB2 SQL query which will be executed by DB2, with the results returned to SAS as either a SAS dataset if you used CREATE TABLE or a PROC PRINT if you didn't.

Pass Through access looks like this:

```
PROC SQL;
  CONNECT TO DB2 (SSID=Q1);
  %PUT &SQLXMSG;
  CREATE TABLE OUT.OCT AS
  SELECT  *
  FROM CONNECTION TO DB2
  (SELECT *
  FROM    DSS.OA0041T_ORDER
  WHERE   ORD_TYPE_IND = 'M'
  );
  %PUT &SQLXMSG;
  DISCONNECT FROM DB2;
  %PUT &SQLXMSG;
```

You issue a CONNECT TO DB2 statement, giving the subsystem id in parentheses. DB2 returns any errors it finds to the automatic variables SQLXMSG and SQLXRC. SQLXMSG contains the return code so you don't need to write them both out. I highly recommend you make a practice of using %PUT to write the error message to the log after both the CONNECT and DISCONNECT statements as well as after the SELECT statement.

The SAS SQL SELECT statement comes first (see below for how to take advantage of this) with FROM CONNECTION TO DB2. The part in parentheses after this will be sent directly to DB2 for parsing and execution. (That is also described in more detail below). Following the close parenthesis and semi-colon for the SELECT statement you DISCONNECT FROM DB2.

The key to using Pass Through access is to remember that the piece inside the parentheses is not parsed by SAS but handed off **as is** to DB2 for execution. Therefore, all the table names and field names inside the parentheses are full DB2 names. The SQL must be DB2 valid, meaning for example no CASE logic or SELECT .... AS. Outside the parentheses you are working in SAS: variable names must be only 8 characters long, and you have the full power of SAS SQL. Thus you can rename variables and perform calculations:

```
PROC SQL;
  CONNECT TO DB2 (SSID=Q1);
  CREATE TABLE OUT.OCT AS
  SELECT   ORD_ID
    ,ORD_MER_ AS SEQNBR
    ,ORD_MER0 * UNIT_PRC AS TOTPRICE
  FROM CONNECTION TO DB2
  (SELECT  ORD.ORD_ID
    ,ORD_MER_SEQ_NBR
    ,ORD_MER_ORD_QTY
    ,UNIT_PRC
  FROM    DSS.OA041T_ORDER     A
  ,DSS.OA021T_ORD_MER   B
  WHERE   A.ORD_ID = B.ORD_ID
    AND A.ORD_TYPE_IND = 'M'
  );
```

You can perform calculations in the DB2 SQL as well, but since you can't rename the results you have to figure out what DB2 will name the new variable. It will be something like EXPRESSN.

Pass Through access works as well as the DB2 SQL query works. This means that if you encounter performance problems, you can work on them with your DBA. You (or they) can run EXPLAINs to determine whether DB2 is using the indexes you think it should be using to optimize the query.

**Not all data in DB2**

But what if not all your data are in DB2 tables? What if, for instance, you want to select records from a DB2 table (or tables) that correspond with a set of key values you have stored in a SAS dataset? Let's take as an example a simple query in which I have a set of customer id numbers in a SAS dataset and I want to find out the address id numbers for these customer ids.

I have several theoretical options for accomplishing what would be a simple merge if both files were already in SAS.

1) SAS dataset and DB2 access descriptor

2) both files in DB2

3) using macros to send key values to DB2

4) both files in SAS

5) simulated access descriptor
   (SAS view using Pass-Through)

**SAS Dataset and DB2 Access Descriptor**

As I have already said, access descriptors don't perform at an acceptable level of efficiency for large files.

Next, we consider loading the SAS key values into DB2 and performing the merge there.

**Writing to DB2**

This option works only under the following conditions:

1) You have write access to the DB2 subsystem in which your lookup table resides, which is not necessarily a given.

2) The large table you want to extract data from is indexed on the key field you want to MERGE on.

If either of these conditions is not met, you shouldn't use this tool. Let's say for the moment, however, that you have met the conditions. Here is how we use this technique.

1) Write a flat file containing the key values.

2) Use a DB2 utility to load from the flat file to a DB2 table.

3) Join the DB2 table of key values with other DB2 table(s) and write result to SAS file.

Although we don't have complete write access to DB2, our DBAs have set up for us several single column tables in the query subsystem for frequently used key fields. In my example, I have 5 million customer ids. I would load these values into a table consisting only of a Customer ID field. Then I would use Pass Through to submit an SQL query joining this table to the lookup table, creating a SAS dataset of the results.

To load the Customer IDs into the single column table, we use a DB2 utility that loads from a flat file. We have found this to work more quickly than a SAS load.

Since several SAS users have access to these loadable tables, it is important to load and read in one job so another user can't load different values in between your load and use of the table.

To illustrate what the final step might look like:

```
PROC SQL;
  CONNECT TO DB2 (SSID = P1);
  CREATE TABLE MATCH AS
  SELECT * FROM
        CONNECTION TO DB2

 (SELECT A.CUST_ID, ADDR_ID
   FROM
   PROD.AA0071T_CUST_IDS A,
  PROD.AA0031T_CUST_ADDR B
   WHERE A.CUST_ID =
            B.CUST_ID);

 DISCONNECT FROM DB2;
```

Note that in DB2 there are no Outer Joins, or unions. When you do a join like this in DB2, the resulting dataset contains only keys that were present in all the tables. Any keys you were looking up that were not found on all the tables you joined with will not be in the output dataset.

**Using Macros**

To review then, we have discarded access descriptors as a technique, and to put your list of SAS keys into DB2 you need write access. That also required that you write a flat file, load it, and do a join. What if you don't have write access or only need to look up a couple of values? The next tool I listed was using macros to pass values to DB2.

The conditions for using the macro technique are:

1) The DB2 table(s) is(are) indexed on the field you want to select on.

2) You have no more than about 10,000 key values to look up.

The macro technique is based on the knowledge that, given a hard coded value of an indexed field, DB2 will return the matching record(s) in a heartbeat. This is something it is really good at.

So, said Paul Kent and Craig Gavin, why not write a macro to turn a SAS dataset of key values into a list of key values that can be inserted in a WHERE clause as part of a Pass Through SQL query.

```
PROC SQL;
  CONNECT TO DB2 (SSID = P1);
  %PUT &SQLXMSG;
  CREATE TABLE MATCH AS
  SELECT * FROM
        CONNECTION TO DB2

  (SELECT CUST_ID, ADDR_ID
    FROM
      PROD.AA0031T_CUST_ADDR
      WHERE CUST_ID IN
  ('12345','23457','49392','23948')
   );
```

Briefly, the macro looks like this.

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

```
DATA _NULL_;
 LENGTH Y $ 20;
 SET SAS.S2000 NOBS = TOTAL
                END = DONE;
 IF _N_ = 1
  THEN CALL SYMPUT
 ('NUMWHERE',PUT(TOTAL, 4.));
 IF NOT DONE
  THEN Y = " ' " ||CUST_ID|| " ' ,";
    ELSE  Y = " ' " ||CUST_ID|| " ' ";
  CALL SYMPUT
    ('WHER'||LEFT(_N_), Y);
RUN;

PROC SQL;
 CONNECT TO DB2 (SSID = P1);
 %PUT &SQLXMSG;
 CREATE TABLE MATCH AS
 SELECT * FROM
      CONNECTION TO DB2

 (SELECT CUST_ID, ADDR_ID
   FROM
    PROD.AA0031T_CUST_ADDR
    WHERE CUST_ID IN
          (%WRITE)
  );
```

This macro is documented in the 1993 SESUG Proceedings in a paper by Craig Gavin.

It works very well, up to a point. That point is the 32k statement size limit. If you have so many key values to look up that your WHERE list exceeds 32k characters, DB2 chokes. So the practical limit of this technique alone is the number of values you can fit in the list, remembering to take into account commas and, if the values are character, quotes.

However, what if you could surround this macro with one that breaks up your SAS dataset of key values into chunks that are of manageable size to this technique?

That's what Tom Finn did, in a macro called PARTDB2 which is shown in Exhibit 1 (clarifying comments courtesy of Jack Shoemaker) and explained in Appendix A. PARTDB2 takes a SAS dataset of key values, breaks it up into chunks of whatever size you specify, and submits queries like the one we just looked at in successsion until it has used up all the values.

It is PARTDB2 that expands the utility of the macro technique from a hard limit of about 2,500 values to whatever your patience will tolerate. It will theoretically work on any volume, but I find it the method of choice only up to about 10,000 values.

Note again that keys not found in the lookup table will not exist on the output dataset.

**Both Files in SAS**

Both techniques I have discussed so far, writing to DB2 and using macros, require that the DB2 lookup table be indexed. What if that is not the case? Or what if you don't have write access and you have more than 10,000 values to look up?

Another tool in our DB2 access kit might seem a bit inefficient at first. That is to pull the lookup table out of DB2 into SAS. Sort of like bringing the mountain to Mohammed. But this tool is necessary, and sometimes preferable to others. In addition to situations where you don't have write access or where DB2 doesn't have the right index, this technique can be used if many small lookups can be performed on the same extract between refreshes or updates.

Conditions for using this technique are not requirements but guidelines. That is to say, you can use this technique under other conditions but it may not be the best.

1) No index on DB2 table

2) No write access and huge list to look up

3) High volume of queries from SAS between updates to DB2 table

If you decide to get your DB2 lookup table into SAS, you have a couple of options. A simple use of Pass Through can make the copy for you. This also allows you to join 2 or more tables to create one SAS dataset containing all the values you need from DB2. But if all you want is a subset of the fields or records from one DB2 table, you should consider a proprietary product called Platinum.

Platinum goes at the VSAM files underlying DB2 and can create flat files from these at lightning speed. It allows you to select fields and records, but not join tables. You then have to read the flat files into SAS, but in a

clocktime pinch it beats even pass through access for high volume reads of DB2.

**Simulated Access Descriptor**

The downsides of bringing your DB2 table out into SAS are obvious: it consumes valuable resources like DASD and even CPU. You'll recall, though, it was the option you should consider if you do not have write access to the DB2 subsystem where the lookup table resides and you need to look up more than about 10,000 values.

Stephen Scott recently brought to my attention one more technique, which I have tested, and it works very well. That is what I am calling the simulated access descriptor.

You can create a SAS View of a DB2 table (or tables) using pass through, which allows DB2 to take advantage of its indexes. Then you can treat that view, like an access descriptor, as if it were a SAS dataset. In the example I'm using today of trying to extract 5 million customer address ids from a DB2 table, I would create the View of the DB2 table then do a SAS DATA step merge of the view with my list of keys.

```
PROC SORT DATA=IN.CUSTIDS
        OUT=CUSTIDS NODUPKEY;
   BY   CUST_ID;
PROC  SQL;
   CONNECT TO DB2 (SSID=DBP1);
   %PUT &SQLXMSG;
   CREATE VIEW DB2ADDRS AS
   SELECT CUST_ID, ADDR_ID
   FROM CONNECTION TO DB2
   (SELECT CUST_ID, ADDR_ID
   FROM PROD.AA004T_CUST_ADDR
   ORDER BY CUST_ID
   );
   %PUT &SQLXMSG;
   DISCONNECT FROM DB2;
   %PUT &SQLXMSG;
DATA CUSTADDR;
   MERGE CUSTIDS(IN=WANTED)
         DB2ADDRS;
   BY CUST_ID;
   IF WANTED;
```

A couple of things to notice here. I'm using the ORDER BY in my pass through instruction knowing that the DB2 table is indexed on CUST_ID, so it can easily supply the data in the correct order for the merge. If the index did not exist, this would not be a good technique to use.

Note also that here you can keep the no hits in your resulting dataset.

This is a simple example. You could create a view of joined tables, select additional fields, select a subset of records, whatever you needed for the situation. The key to deciding whether to do subsets and selects in DB2 or in SAS is to understand which can do it faster. Sometimes this takes testing and experimentation.

**Summary**

I have described several techniques for making data stored in DB2 accessible to SAS. To review the techniques and when to use them:

1) Don't use Access Descriptors on large tables

2) If all the information you want is in DB2, and you want to use SAS to manipulate it, use Pass Through access to submit a DB2 query and bring the results back as a SAS dataset.

The rest of the techniques apply when you have some data in SAS and some in DB2 and you want to bring the two together.

3) The first aspect of the situation to check is whether your DB2 table is indexed. If your DB2 table is not indexed, you will probably get better performance by using Platinum to dump the data to a flat file, read it into SAS and go from there. If you have to do this frequently to the same table you may want to consider asking your DBA's to add the index.

4) If your DB2 table is indexed, your next consideration is how many records you want to extract. If you have a list of key values in a SAS dataset, and it is only a handful, say less than a thousand, you will probably want to use the macro technique to pass the values to DB2 in a list and extract the matching records.

5) If you have more than a handful of keys, the choice of technique depends on whether you can load the keys into DB2. If you can load the keys into DB2, that technique works well.

6) If you can't load your list into DB2, then you go back and consider the volume again. If it is less than 10 thousand, you can use the PARTDB2 macro to break the volume down into manageable lists and extract the records that way.

7) Finally, if you can't load your SAS dataset of keys into DB2 and you have more than 10 thousand key values to extract, a SAS View of DB2, or simulated access descriptor will probably be your best choice.

There are probably more options than this to consider, but I have found this tool kit invaluable in using SAS with DB2 in a variety of situations.

### References

Kent, Paul [1992] "Fun with the SQL Procedure--An Advanced Tutorial" pp 167-175 *SUGI 17 Proceedings*

Gavin, Craig James [1993] "SAS/ACCESS Techniques for Large DB2 Databases" pp 99-105 *SESUG 93 Proceedings*

Loren, Judy and Dickson, Alan [1994] "Processing Large SAS and DB2 Files: Close Encounters of the Colossal Kind" pp 1497-1503 *SUGI 19 Proceedings*

Loren, Judy and Shoemaker, Jack [1995] "Retrieving Data from Large DB2® Tables Based on Keys in SAS® : The Sequel" pp 57-66 *NESUG 95 Proceedings*

Scott, Stephen [1993] "Why We Replaced DB2 Software with SAS Software as a Relational DBMS for a 30-Gigabyte User Information System" pp 187-196 *SUGI 18 Proceedings*

### Trademarks



### For Further Information

I welcome comments by e-mail. You can reach me at 102037.3235@compuserve.com

## Appendix A--The PARTDB2 Macro

The PARTDB2 macro described in this paper allows you to break up your list of keys into pieces of any size. It then executes multiple queries, putting the number of keys you specified into each WHERE clause and looping until the list of keys is exhausted. When you're done you have one SAS dataset with the results from all the queries appended.

### An Example

To illustrate the points in this paper, we can use the example of looking up information about customers in a DB2 database. Suppose we have customer id numbers and need to look up name, address and phone number. With one id number the code might look like this:

```
PROC SQL;
CONNECT TO DB2 (SSID=DBQ2);

SELECT * FROM CONNECTION TO DB2
(SELECT *
 FROM PREF.CUST_ADDR ADDR,
      PREF.CUST_NAME NAME,
      PREF.CUST_PHONE PHONE
  WHERE ADDR.CUST_ID = '12345'
    AND NAME.CUST_ID = ADDR.CUST_ID
    AND PHONE.CUST_ID = ADDR.CUST_ID
  );
DISCONNECT FROM DB2;
```

With more than one id number, the

```
WHERE ADDR.CUST_ID =
```

would change to

```
WHERE ADDR.CUST_ID IN (...)
```

To save you typing the list of keys you want in the parentheses, you can use a macro (discussed above) to generate this list automatically from a SAS dataset containing the values you want. Supposing this macro is called WRITE, the WHERE clause would look like:

```
WHERE ADDR.CUST_ID IN (%WRITE)
```

It's when this list gets too long for DB2 to contemplate that you need PARTDB2.

### How to Use the PARTDB2 Macro

Exhibit 1 at the end of this paper shows the actual code for the PARTDB2 macro.

PARTDB2 assumes you have a list of keys in a SAS dataset. It requires the following parameters:

**Incoming Data Set (IDS=)**

You can use a two-level name to point to a stored dataset, or you can read in a WORK dataset coming from a previous DATA or PROC step.

**Output Data Set (ODS=)**

Tell PARTDB2 where you want to put the result of the queries.

**How many keys at a time (N=)**

You can experiment with the number you get best results with. The limit depends on the number of bytes in your key values and whether they are character (requiring a set of quotes per value) or numeric.

**The name of the SAS key variable (SASVAR=)**

PARTDB2 will take the values of this variable and put them into the WHERE clause. **No other variables from the incoming dataset can be carried through to the output dataset.**

**The type of the SAS key variable (TYPE=)**

CHAR(acter) or NUMeric. (Actually the macro just looks for CHAR; if the value of TYPE is not CHAR, it assumes numeric.)

**The format of the SAS key variable (VARFMT=)**

The format only matters if the key variable is numeric. It uses the format to put the numeric values into the WHERE clause. Character values are put in default format with single quotes around them.

**The name of the DB2 key field (DB2VAR=)**

This is the field PARTDB2 uses in the WHERE clause to compare to the SAS key values. Note that if this field name appears in more than one table in your FROM list, you must specify a prefix (table name or alias) to decide which table's field will be used in the WHERE clause. You can join the two DB2 tables in the WHERE parameter (below).

**The DB2 subsystem your table resides in (SSID=)**

This is the value that will be supplied in the CONNECT TO DB2 (SSID=value) statement. No, even PARTDB2 can't access more than one subsystem at a time.

**Your SELECT statement (SELECT=%STR( ) )**

Note that the list of fields you want to extract from DB2 should be enclosed in a %STR( ) function. Note also that this is the DB2 SELECT and should contain full DB2 field names.

**Your FROM statement (FROM=%STR( ) )**

Here's where you identify the fully qualified DB2 table name(s) containing the fields you want to extract. When you have more than one table in this list, aliases help in identifying the field sources.

**Renaming the DB2 fields for SAS (AS=%STR( ) )**

This parameter allows you to rename the DB2 fields as they come into the resulting SAS dataset (ODS, above). It operates positionally with the SELECT parameter; the first field in the SELECT parameter receives the first variable name in the AS list, the second field receives the second variable name, etc. This avoids you having to figure out which tie-breaker was used for the DB2 names that match for the first 8 characters.

**Additional WHERE restrictions (WHERE=%STR( ))**

The match of the SAS key variable with one DB2 field is taken care of for you. This parameter is for specifying additional restrictions or joining criteria, especially needed if you are accessing more than one table.

**Restarting (FO=)**

If the macro fails in the middle, the results of completed queries remain saved to the output dataset. You can use FO= (stands for FIRSTOBS=) to tell PARTDB2 where to start (which observation to start with) in the incoming dataset.

**Appending results to an existing dataset (FIRSTACT=)**

Independently of restarting, you decide whether you want to create a new dataset with this execution of PARTDB2 or append to an existing dataset. FIRSTACT=CREATE will cause the FIRST query to use CREATE TABLE AS; all subsequent queries in that execution of PARTDB2 will use INSERT INTO. If you specify FIRSTACT=INSERT, even the first query will use INSERT INTO.

**Controlling the number of loops (NLOOPS=)**

You can limit the number of queries PARTDB2 initiates by coding a number here. You would probably be interested in the TRACK= parameter below.

**Keeping TRACK of where you are (TRACK=)**

Here you specify the name of a dataset that will store the observation number you should start with if you want to complete the list of keys after either a failure part way through the list or a stop caused by hitting the NLOOPS limit you specified.

**An Example**

To illustrate the use of the parameters, suppose as in our previous example we have a list of customer ids and wish to retrieve name, address and phone numbers for each customer. This time, our list of desired id numbers resides in a SAS dataset called WANTED with a variable name of CID. Using PARTDB2, we would code:

```
//MAC DD DISP=SHR,DSN=MACRO.LIB
//IN DD DISP=OLD,DSN=SAS.WANTED

OPTIONS NOMPRINT     SASAUTOS=(MAC) OBS=MAX;
%PARTDB2(IDS=IN.WANTED
  ,ODS=IN.ADDRS
  ,N=1000
  ,SASVAR=CID
  ,DB2VAR=A.CUST_ID
  ,SELECT=%STR(A.CUST_ID, CITY, STATE,
    ADD_LINE1, ADD_LINE2, ZIP_CODE, NAME,
    PHONE_NUMBER, AREA_CODE)
  ,AS  = %STR(CID, CITY, STATE, ADDR1, ADDR2,
    ZIP, NAME, PHONE, AREACODE)
  ,FROM=%STR(   PREF.CUST_ADDR A,
                PREF.CUST_PHONE B,
                PREF.CUST_NAME C)
  ,WHERE=%STR(A.CUST_ID = B.CUST_ID
         AND A.CUST_ID = C.CUST_ID)
  ,SSID=DBP1
  );
```

Note that some parameters are not required at all unless you want to take advantage of a particular feature. Other parameters can be allowed to default, such as the TYPE=CHAR, if the default suits your application.

**Defaults**

You can establish the PARTDB2 macro at your site with whatever defaults you find most useful. Although it is not shown this way in Exhibit 1, you can even use default values for DB2VAR, FROM, SASVAR, ODS, etc.

**Suggestions for use**

Because of the volume of log generated, it is best to specify the NOMPRINT option. To further limit the lines of feedback, also use NOFULLSTATS and NONOTES.

The values of the FROM, WHERE, SELECT, and AS parameters should be enclosed in the %STR(). For more complex strings, subqueries for example, you may have to resort to %NRBQUOTE() (no rescan blind quote) or another macro as described below.

(The PARTDB2 macro code follows this page.)

Exhibit 1. The PARTDB2 Macro
Author: Tom Finn
Comments: Jack Shoemaker

```
%macro PARTDB2(
      IDS=,          /* Input dataset */
      ODS=,          /* Output dataset */

      FROM=,         /* DB2 table name */
      SSID=DBQ1,     /* DB2 SSID */

      DB2VAR=,       /* Key var name in &FROM DB2 table */
      SASVAR=,       /* Key var name in &IDS dataset */
      TYPE=CHAR,     /* Key var type */
      VARFMT=BEST.,/* Format to write numeric keys */

      SELECT=,       /* DB2 select statement */
      AS=,           /* SAS alias names */

      /* Conjunctive DB2 where clause fragments */
      WHERE=,        /* As a literal */
      WHEREMAC=,     /* As a SAS macro */

      TRACK=,        /* Tracking dataset name */
      FO=1,          /* First obs in &IDS */
      FIRSTACT=CREATE, /* First action on &ODS */

      N=400,         /* Number of key vals in each loop */
      NLOOPS=,       /* Max number of loops */
      INOBS=);       /* SAS PROC SQL inobs= value */

    /* Initialize and scope macro variables       */
    %local NOBS I STOPRC V1 - V&N;
    %let TYPE = %upcase( &TYPE );
    %let STOPRC = 0;

    /* Write restart message to SAS log           */
    %if &FO ne 1 %then
      %put ******* RESTARTING *********;

    /* Find number of obs in &IDS and place       */
    /*   value in macro variable &NOBS            */
    data _null_;
      if 0 then set &IDS nobs = nobs;
      put nobs=;
      call symput( 'NOBS',
        left( put( nobs, best. ) ) );
      stop;
      run;

    /* Write &FO and &NOBS values to SAS log      */
    %put FO=&FO NOBS=&NOBS;

    /* If &IDS has no observations, then look     */
    /*   for records with missing key values      */
    %if &NOBS eq 0 %then %do;

      proc sql
      %if %length( &INOBS ) gt 0 %then
        inobs = &INOBS;
        ;

      connect to db2( ssid = &SSID );

      create table &ODS as
        select * from connection to db2(
          select &SELECT
          from   &FROM
          where
            %if &TYPE eq CHAR
              %then   ( &DB2VAR = ' ' );
            %else     ( &DB2VAR = . );
            %if %length( &WHERE ) GT 0
              %then   and &WHERE ;
            %if %length( &WHEREMAC ) GT 0
              %then   and %&WHEREMAC ;
        ) as a( &as );

      %put &SQLXRC &SQLXMSG;

      quit;

      data &ODS;
        set &ODS;
        stop;

    %end;

    /* Otherwise, build &ODS by looping through   */
    /*   &__FROM using &N keys at a time          */
    %else %do %while(
      &FO le &NOBS and
      &STOPRC eq 0 and
      X&NLOOPS ne X0 );

      /* Write &FO value to &TRACK if requested   */
      %if %length( &TRACK ) gt 0 %then %do;
        data &TRACK( keep = fo );
          fo = &FO;
          if _n_ = 1 then output;
      %end;

      /* Otherwise just start a _NULL_ data step */
      %else %do;
        data _null_;
```

```
%end;

/* Read in &IDS starting at &FO            */
/* Put values of &SASVAR into              */
/*    macro variables &V1 to &V&N          */
set &IDS(
  keep = &SASVAR
  firstobs = &FO ) end = end;
length nn $7.;
nn = left( put( _n_, best. ) );
%if &TYPE eq CHAR %then %do;
  call symput( 'V' || nn,
    "'" || &SASVAR || "'" );
%end;
%else %do;
 call symput( 'V' || nn,
   left( put( &SASVAR, &VARFMT ) ) );
%end;


if end or _n_ ge &N then do;
  call symput( 'N' , nn );
  stop;
end;
run;

proc sql
%if %length( &INOBS ) gt 0 %then
  inobs = &INOBS;
;

connect to db2( ssid = &SSID );

%if &FIRSTACT eq CREATE %then
  create table &ODS as;
%else
  insert into &ODS;
%let FIRSTACT = INSERT;

select * from connection to db2(
  select &SELECT
  from   &FROM
  where  &DB2VAR in ( &V1
  %do I = 2 %to &N;
    , &&V&I
  %end;            )
  %if %length( &WHERE ) GT 0
    %then and &WHERE ;
  %if %length( &WHEREMAC ) GT 0
    %then and %&WHEREMAC ;
  ) as a( &as );

%put &SQLXRC &SQLXMSG;
    %if &SQLXRC ne 0 %then %let STOPRC = &SQLXRC;
    quit;

    %let FO = %eval( &FO + &N );
    %if %length( &NLOOPS ) gt 0 %then
      %let NLOOPS = %eval( &NLOOPS - 1 );

  %end;

  /* Create tracking dataset if requested      */
  %if X&NLOOPS eq X0 and
      %length( &TRACK ) gt 0
    %then %do;
      data &TRACK( keep = fo );
        fo = &FO;
        output;
        stop;
  %end;

  /* Abort program and pass SQL RC back        */
  %if &STOPRC ne 0 %then %do;
    data _null_;
    %if &STOPRC gt 0 %then abort &STOPRC ;
    %else abort %eval( -1 * &STOPRC );
    ;
  %end;

%mend PARTDB2;
```