Paper 127-26

Joining SAS® and DBMS Tables Efficiently

Garth W. Helf, IBM Corporation, San Jose, CA

ABSTRACT

A common task in Data Warehouse applications is joining a SAS data set with tables in a relational database (DBMS), such as DB2, Oracle, or Teradata. For example, you work for a company that manufactures televisions, you have a SAS data set that contains serial numbers of 100 televisions returned for warranty repair, and you want to look up information for these 100 televisions in a DB2 table that contains the manufacturing history of all 1 million TVs you made last year. SAS versions 6.12, 7, and 8 provide tools in the SAS/ACCESS® Software for Relational Databases product to transparently join SAS data sets with DBMS tables in the SQL procedure or DATA steps. This paper, intended for intermediate and advanced SAS users, describes these tools, explains why performance can be very poor when you join a SAS data set with a large DBMS table, and presents some solutions to make the join more efficient. One solution is to use the data set options DBINDEX= and DBKEY= in the SAS/ACCESS LIBNAME Statement, which are new in Versions 7 and 8. Another solution is the %DBMSlist macro, which passes SAS data set values in chunks to the SQL Procedure Pass-Through Facility.

INTRODUCTION

There are many times when you want to write a query to a DBMS table and return data only for the values of a variable in a SAS data set. For example, you have a flat file of serial numbers from which you create a SAS data set with INFILE and INPUT statements in a DATA step. Or maybe you created a SAS data set from a query against a DBMS table at another location in your company, and now you want to query a DBMS table at your location only for the key values in your SAS data set. SAS provides tools for joining a SAS data set with a DBMS table.

PROC SQL CAN JOIN SAS AND DBMS TABLES, BUT BE CAREFUL!

SAS versions 6.12, 7, and 8 can all join a SAS table to a DBMS table in a PROC SQL step. For example, suppose we have a SAS data set called Warranty that contains the serial numbers (variable sn) for 100 televisions that were returned for warranty service, and we want to join it with a DBMS table called MfgHist which contains the manufacturing history for all one million TV sets made so far. The following PROC SQL step would work:

```
Proc SQL;
 create table History as
 select * from Warranty a, dbms.MfgHist b
 where a.sn=b.sn;
 quit;
```

In SAS Version 6.12, dbms.MfgHist refers to a view descriptor, which is associated with an access descriptor. A discussion of access and view descriptors is beyond the scope of this paper. In Version 7 and 8, dbms.MfgHist refers to a SAS/ACCESS LIBNAME data set association, which is described in a later section.

What's wrong with this PROC SQL approach to join a SAS data set with a DBMS table? Nothing, if your DBMS table is fairly small. However, when the DBMS table is large, this PROC SQL step is very inefficient because every row in the DBMS table is returned to your SAS session and joined with your SAS data set by PROC SQL. In our TV warranty example, all one million rows from the MfgHist DB2 table will be returned to your SAS session, and all but 100 rows will be discarded because they don't match the serial numbers in our SAS data set.

There are two good ways to solve this problem. One solution for versions 7 and 8 only is to use the DBKEY= and DBINDEX= data set options with the SAS/ACCESS LIBNAME statement. Another is to write a macro that passes the values in your SAS data set in groups to an SQL Procedure Pass-Through Facility query. Such a macro can be used with version 6.12 as well as versions 7 and 8. This paper includes one such macro called %DBMSlist that I use extensively.

SAS/ACCESS LIBNAME STATEMENT

Starting with Version 7, SAS provides a much better way to access DBMS tables than access and view descriptors, through a new feature called the SAS/ACCESS LIBNAME statement. This statement allows you to assign a SAS libref directly to tables in a DBMS. For example, if our TV table is in a DB2 database called TVDB, a SAS/ACCESS libname statement would look like:

```
libname dbms db2 dsn=tvdb uid=helf pwd=mypw;
```

After this LIBNAME statement is issued, you can access DB2 tables in PROC and DATA steps by refering to data set dbms.*DB2_table_name*.

There are system options, SAS/ACCESS LIBNAME statement options, and data set options that monitor and affect the efficiency of SAS steps that join SAS and DBMS tables. This section describes the SASTRACE system option, and the DBKEY= and DBINDEX= data set options.

SASTRACE: A GREAT OPTION FOR DEBUGGING SAS/ACCESS LIBNAME STATEMENTS

How did I know that SAS retrieves an entire DBMS table in certain situations? There is a fantastic system option called SASTRACE that displays detailed information about the commands that SAS/ACCESS sends to your DBMS. This option is incredibly useful when you are debugging a program that is using a SAS/ACCESS Libname statement. The syntax for this option is:

```
options sastrace=',,,d' sastraceloc=saslog;
```

The SASTRACE option turns on detailed DBMS messages, and the SASTRACELOC option tells SAS where to write the messages, in this case the SAS log. For example, when you submit a LIBNAME statement to assign a libref to a DB2 database, you will see the following messages in the SAS log:

```
TRACE: Successful connection made, connection id
     0 0 1296057922 no_name 0 Submit
TRACE: Database/data source: tfdiskdb 1
     1296057922 no_name 0 Submit
TRACE: USER=HELF, PASS=XXXXXXX 2 1296057922
     no_name 0 Submit
TRACE: AUTOCOMMIT is NO for connection 0 3
     1296057922 no_name 0 Submit
19    libname db2sys db2 &tfdiskdb schema=db2sys;
NOTE: Libref DB2SYS was successfully assigned as
     follows:
       Engine:        DB2
       Physical Name: tfdiskdb
```

Each line in the log that starts with TRACE: is a message about how SAS interacts with the DBMS. Further examples of trace output is shown in later sections. This trace information in the log is very useful when you need to talk with SAS Technical Support, and when you want to talk to your database administrator. The administrator of your Oracle or DB2 database probably knows nothing about SAS, but the SASTRACE information in the log will be quite helpful for him or her to debug a database problem.

Hint: I could not find this anywhere in the documentation, but to turn off tracing, submit the following statement:

```
   options sastrace=',,,';
```

DBKEY= DATA SET OPTION

The DBKEY= option lets you specify the column(s) in the DBMS table to use as a serach key. An actual index on this column in the DBMS table is not required, and SAS does not attempt to determine if one exists. It is used like this:

```
   libname dbms db2 dsn=tvdb uid=helf pwd=mypw;
   proc SQL;
    create table History as
    select * from Warranty a,
      dbms.MfgHist (dbkey=sn) b
    where a.sn=b.sn;
    quit;
   libname dbms clear;
```

In this example, the DBKEY= option instructs the SQL procedure to pass the WHERE clause to the SAS/ACCESS engine in a form similar to WHERE SN=*host-variable*. The engine then passes this optimized query to the DBMS server. The host-variable is substituted, one at a time, with SN values from the observations in the SAS data set Warranty. As a result, only rows that match the WHERE clause are retrieved from the DBMS. Without this option, PROC SQL retrieves all the rows from the MfgHist table.

Here are some examples of the DBKEY= option. In these exmaples, data set LOTS contains a variable called LOT whose values will be used as a key to retrieve data from DB2 table db2sys.disc_stat_history. I know that column LOT in this DBMS table has an index.

Example 1 - Single Key Variable: This first example shows a simple query with one variable as the search key:

```
   proc sql;
    create table lot_data3 as
    select a.lot, a.trdate, a.trtime, a.resource
    from db2sys.disc_stat_history (dbkey=lot) a,
       lots b
    where a.lot=b.lot;
    quit;
```

```
TRACE: Using FETCH for file DISC_STAT_HISTORY on
     connection 0 6 1296151584 no_name 0 SQL
TRACE: Change AUTOCOMMIT to YES for connection
     id 0 7 1296151584 no_name 0 SQL
TRACE: SQL stmt prepared on statement 0,
     connection 0 is:  SELECT * FROM
     db2sys.DISC_STAT_HISTORY 8 1296151584
     no_name 0 SQL
TRACE: DESCRIBE on statement 0, connection 0. 9
     1296151584 no_name 0 SQL
TRACE: SQL stmt prepared on statement 0,
     connection 0 is:  SELECT  LOT, TRDATE,
     TRTIME, RESOURCE  FROM
     db2sys.DISC_STAT_HISTORY  WHERE (((lot= ? )
     OR ((lot IS NULL ) AND ( CAST(? AS LONG
     VARCHAR) IS NULL )))) FOR READ ONLY  10
     1296151584 no_name 0 SQL
TRACE: Open Cursor with new index value 12
     1296151584 no_name 0 SQL
TRACE: Close cursor from statement 0 on
     connection 0 13 1296151584 no_name 0 SQL
```

Note that SAS first prepares a SELECT * statement and then does a DESCRIBE to find the names and attributes of the DBMS columns named in the query. Then, SAS prepares a statement to actually retrieve data and constructs a WHERE condition of LOT=?. SAS then uses OPEN CURSOR and CLOSE CURSOR statements to pass each value of variable LOT in the SAS data set to the DBMS and return the results to SAS.

The performance of queries using the DBKEY= options is usually quite good. In this example, my LOTS dataset had 200 rows and it took 3 seconds to return the data from the DBMS table that had 120K rows. Without the DBKEY= option, SAS retrieves all of the rows in the DBMS table and applies the WHERE clause in SAS. When I removed the DBKEY= option from this query, it took 41 seconds to complete. I also ran a similar query with the DBKEY= option against a much larger table, about 5M rows, and this query took only 4 seconds.

However, I have seen cases where a PROC SQL query with the DBKEY= option takes 10 times longer than using the macro %DBMSlist described in the next section. This generally happens when the DBMS server is at a remote location with a low bandwidth network connection. The message here is that you need to carefully evaluate different methods for running queries that join SAS and DBMS tables.

Example 2 - Multiple Key Variables: This example shows a query where multiple variables in the SAS data set are used to form the search key. Note that you must enclose two or more column names in parentheses after the DBKEY= token. To save space, I will list only the SASTRACE statement that shows the statement SAS sends to the DBMS.

```
   proc sql;
    create table lot_data as
    select a.lot, a.trdate, a.trtime, a.resource
    from db2sys.disc_stat_history (dbkey=(lot
       trdate)) a, lots b
    where a.lot=b.lot and a.trdate=b.trdate;
    quit;
```

```
TRACE: SQL stmt prepared on statement 0,
     connection 0 is:  SELECT  LOT, TRDATE,
     TRTIME, RESOURCE  FROM
     db2sys.DISC_STAT_HISTORY  WHERE (((lot= ? )
     OR ((lot IS NULL ) AND ( CAST(? AS LONG
     VARCHAR) IS NULL ))) AND ((trdate= ? ) OR
     ((trdate IS NULL ) AND (CAST(? AS
     VARCHAR(50))  IS NULL )))) FOR READ ONLY  4
     1296167379 no_name 0 SQL
```

It appears that PROC SQL with the DBKEY= option will only create WHERE clauses when the join condition is equality, for example A.LOT=B.LOT. If you want to use other SQL clauses like BETWEEN or greater than, use SASTRACE and make sure SAS is building an SQL statement that is efficient for your DBMS.

Example 3 - DBKEY= Option in a DATA step: You can also use the DBKEY= data set option in a DATA step by using the KEY=DBKEY option of the SET statement. This example uses a data step to join the SAS data set with a DBMS table:

```
   data lot_data;
    set lots;
    set db2sys.disc_stat_history (dbkey=lot
      keep=lot trdate trtime resource)
      key=dbkey;
    run;
```

```
TRACE: SQL stmt prepared on statement 0,
     connection 0 is:  SELECT  TRDATE, TRTIME,
     LOT, RESOURCE  FROM db2sys.DISC_STAT_HISTORY
     WHERE (((lot= ? ) OR ((lot IS NULL ) AND (
     CAST(? AS LONG VARCHAR) IS NULL )))) FOR
     READ ONLY
```

WARNING! Be very careful with this DATA step method when the DBMS table may contain multiple rows for each value of your key variable. This DATA step method returns only one row for each value of the key variable, and the row returned may not even be the same if you rerun the query! For example, when I had 10 values of key variable LOT in the LOTS data set, the PROC SQL method in Example 1 returned 202 rows from the DB2 table (because many transactions are performed on each lot during our manufacturing process), but the DATA step method above returned only 10 rows. Also, queries to a DBMS table do not return data in any particular order unless you use the ORDER

BY clause. Therefore, I saw cases where several rows in the result set were different when I reran the DATA step compared to the first time. The message here is that even though this DATA step method is described in the SAS documentation and supported by SAS Institute, it may be safer to always use the PROC SQL method instead.

Example 4 - DBNULLKEYS= Data Set Option: In Example 1 we saw that SAS builds a WHERE expression like this when you use the DBKEY= option:

```
WHERE (((lot= ? ) OR ((lot IS NULL ) AND (
     CAST(? AS LONG VARCHAR) IS NULL ))))
```

You can see that the WHERE clause is actually two clauses separated by the OR operator. The first clause is for non-null values of your key variable. The second clause is for null values of the key variable. A compound clause like this is less efficient for your DBMS to process than a simple clause, so SAS provides another data set option called DBNULLKEYS= that tells PROC SQL to create the non-null WHERE clause only. If you know that your key variable does not contain null values, you should use this option. The syntax for this option and the WHERE clause created look like this:

```
   proc sql;
    create table lot_data as
    select a.lot, a.trdate, a.trtime, a.resource
   from db2sys.disc_stat_history (dbkey=lot
    dbnullkeys=no) a, lots b
    where a.lot=b.lot;
    quit;
```

```
WHERE ((lot= ? ))
```

Unfortunately, I cannot show you an actual TRACE statement for this option because my SAS 8.1 system and SAS/ACCESS for DB2 says that DBNULLKEYS= is an invalid option. Unpublished documentation I received from SAS Technical Support shows this option in an example using Oracle, so either this option is specific to Oracle or it was introduced in Version 8.2.

DBINDEX= DATA SET OPTION

There is another data set option described in the SAS documentation called DBINDEX= that can be used to improve efficiency of PROC SQL queries that join SAS and DBMS tables. Unfortunately, my experience with this option does not match the documentation. The documentation for DBINDEX= shows that you use this feature to tell SAS to query the DBMS to find indexes on the DBMS table. SAS then attempts to use the indexes on the DBMS table to improve performance. However, my experience is that SAS constructs the same query no matter what value you use with the DBINDEX= option, and that the query SAS constructs depends only on the number of observations in your key values SAS data set.

Example 5 - Small Key Values Data Set: When my SAS data set contains between 1 and 200 observations, SAS constructs a WHERE statement that has an IN list of the key values. In this example, my key values data set has 10 observations:

```
   proc sql;
    create table lot_data as
    select a.lot, a.trdate, a.trtime, a.resource
    from db2sys.disc_stat_history (dbindex=yes)
      a, lots b
    where a.lot=b.lot;
    quit;
```

```
TRACE: SQL stmt prepared on statement 0,
     connection 0 is:  SELECT  LOT, TRDATE,
     TRTIME, RESOURCE  FROM
     db2sys.DISC_STAT_HISTORY  WHERE  ( ( LOT IN
     ( 'DI00036136' , 'DI00037986' , 'DI00038082'
     , 'DI00038359' , 'DI00038836' , 'DI00038915'
     , 'DI00038929' , 'DI00039471' , 'DI00039476'
     , 'DI00039480' ) ) ) FOR READ ONLY  611
     1296429914 no_name 0 SQL
```

Example 6 - Large Key Values Data Set: When my SAS data set contains 201 or more observations, SAS creates an SQL statement to return all rows from the DBMS table and processes the join in SAS! This is not what you want SAS to do if your DBMS table is large. Here is the TRACE output for the PROC SQL step from Example 5, but with 201 observations in my key values data set:

```
TRACE: SQL stmt prepared on statement 0,
     connection 0 is:  SELECT  LOT, TRDATE,
     TRTIME, RESOURCE  FROM
     db2sys.DISC_STAT_HISTORY  FOR READ ONLY  617
     1296430260 no_name 0 SQL
```

This query took 55 seconds to run because it brought all 200K rows from the DBMS table into SAS. With 200 observations in my key values dataset, the query ran in 8 seconds because it submitted a WHERE statement to the DBMS. The message here is be sure to use SASTRACE to test your query to see if the DBINDEX= option does what you want. It may work differently with different DBMS systems.

MACRO %DBMSLIST GENERATES PROC SQL PASS-THROUGH QUERIES

Most of the function of macros like %DBMSlist has been replaced by the more elegant techniques just described in the SAS/ACCESS LIBNAME statement in versions 7 and 8.

The SQL Procedure Pass-Through Facility was an enhancement to the SQL procedure starting in Version 6 that enables you to send DBMS-specific SQL statements to a DBMS server and retrieve DBMS data directly to SAS. Pass-through queries can take advantage of indexes on DBMS columns to process a query more quickly and efficiently. For example, consider the following query to return data from our MfgHist DB2 table for these three TV serial numbers:

```
   proc sql;
    connect to db2 (dsn=tvdb uid=helf pwd=mypw);
    create table History as select * from
        connection to db2 (
      select * from MfgHist where sn in
        (’CGN213’,’SDA980’,’WEF765’)
      for fetch only);
    disconnect from db2;
    Quit;
```

This is called a Pass-Through query because the entire query within the outermost parentheses in the Create Table statement is passed through to the DBMS server unaltered, except for resolution of any macro variables that might be present. If the column SN in table MfgHist has an index, this step should run in a few seconds or less, even though there are a million rows in the table.

The trick to writing a macro that generates SQL Pass-Through queries is to format the key values in your SAS data set into an appropriate WHERE clause for your DBMS. In our TV example, the TV serial numbers are defined as character strings in the DB2 table, so the values of variable SN in the SAS data set must be enclosed in single quotes and separated by commas. You might also want your macro to handle key values that are defined as other data types in the DBMS table, such as numeric, date, time, or timestamp. A more complex macro would handle requests based on multiple key columns, for example a serial number and a manufacturing operation number. The %DBMSlist macro discussed next has all of these features.

Macro %DBMSlist has several arguments: the name of the SAS data set your key values are in, the name(s) of the variable that contain the values you want to pass to the DBMS, the type of variables they are, the name of the SAS data set you want to create, and the query you want to run. The macro builds your key values into a list and assigns it to a macro variable, then runs your query and puts the data from the DBMS into the SAS data set you specified. The syntax of macro %DBMSlist is:

```
    %DBMSlist(dsn, column, vtype, newdsn, dbname,
      query, bitesize=200, test=no, dlm=%str(#));
```

These arguments to macro %DBMSlist are described in detail in Table 1.

Table 1: Syntax of the %DBMSlist Macro

Parameter	Meaning
dsn	Name of the existing SAS data set that contains the values you want to include in a DBMS query. May be a temporary data set (with a one-part name) or a permanent data set (with a two-part name).
column	Name of the variable (or variables) in the data set named in **dsn** that contains the values you want to submit to the DBMS. If more than one variable name, **column** must be in the form of an SQL template enclosed in the **%str** function, with variable names delimited by a special character (the # symbol by default). Refer to Table 2 for more information.
vtype	Type of variable(s) named in **column**, must have one value for each variable named in **column**. Not case sensitive. Currently, 5 types are supported: • **c** - for character data, values are enclosed in single quotes • **n** - for numeric data • **d** - for date data, values are put in DB2 query format, like '1998-11-22'. • **t** - for time data, values are put in DB2 query format, like '10.32.45'. • **dt** - for timestamp data, values are put in DB2 query format, like '1998-10-11-21.34.54.234242'
newdsn	Name of the new SAS data set you want the macro to create. It may be either temporary or permanent.
dbname	Connection information appropriate for your DBMS. For DB2, it can be: 1) The word **prompt**, in which you will be prompted for userid and password 2) Explicit connection information, like **%str(dsn=engdb uid=helf pwd=mypw)** 3) A macro variable that contains your DBMS connection information, like **&engdb**. These macro variables should be placed in your autoexec.sas file for security and maintenance reasons.
query	The query you want the DBMS to process. Put the macro variable **&mylist** in your WHERE statement to specify your list of key values. The query must be enclosed in the macro function **%nrstr**.
bitesize	Optional parameter, default value is **200**. Only this many serial numbers are passed to the DBMS at a time. Make this value smaller if the query takes too long or you get any errors from the DBMS. You can make it larger if you have a large list and the query runs OK.
test	Optional parameter, default value is **no**. If it is anything else, like **yes** or **Yes** or **YES**, the macro passes the first **bitesize** values to the DBMS and does **not** create the output data set. I use this for testing to see how long the query will take with different values of **bitesize**.
dlm	Optional parameter, default value is the pound symbol (#). Defines the delimiter to be used for marking variables in the SQL template. To enter a different character, use the %str function, for example **%str(@)** to use the @ symbol as a delimiter.

Argument COLUMN of Macro %DBMSlist takes different forms depending on whether one variable or more than one variable in the SAS data set forms the key for the DBMS table query. This argument is described in Table 2.

Table 2: Argument COLUMN for Macro %DBMSlist:

Number of key variables	Value of &COLUMN	Contents of &mylist and Where statement syntax

1 variable name	Single name of the variable that contains the value to be passed to the DBMS, for example:
	
	`file`
		&mylist contains values of the variable you specified, separated by commas, like:
		`'13001111HK', '13002222HK', '13003333HK'`
		
		Use a Where statement like:
		`where file in (&mylist)`

More than one variable name	An SQL template enclosed in the %str function that consists of static text and variable names. Each variable name must be delimited at start and end by a special character (# by default). Example:
	`%str(wafer=#mywaf# and row=#myrow#)`
		&mylist contains the SQL template you specified, with variable names substituted with the values of those variables, enclosed in parentheses and separated by the word OR, like:
		`(wafer=258053 and row=33) or (wafer=780784 and row=12)`
		
		Use a Where statement like:
		`where (&mylist)`

Note about Macros: There are several ways to make these macros visible to your SAS program. The easiest way is to set the SASAUTOS option to point to the directory where you put them, like this:

```
   options sasautos=(sasautos,'C:\MYSAS\MACRO');
```

You need to run this only once in each session. Better yet, add it to your AUTOEXEC.SAS file so it gets run every time you start SAS.

You might see a warning message about string length. I've noticed a warning message in the SAS 6.12 log about concatenated string length longer than 200 characters, but the macro still runs fine. I don't get this message in SAS 7 or 8.

Example 7 - Single Character Key Variables: This macro call takes a list of serial numbers in variable SHIPCASS in SAS data set PACKPREP and creates a new SAS data set called DISKSHIP that contains the data returned from the ENGDB database. The DBMS connection information was previously assigned to macro variable &ENGDB.

```
   %DBMSlist(packprep, shipcass, c, diskship,
   &engdb, %nrstr(
    select cassette, trdate, trtime, pwrpak
    from d.current_status
    where cassette in (&mylist) ));
```

Example 8 - Multiple Numeric Key Variables: This macro call takes a list of serial numbers in variables WAFER and ROW in SAS data set MYDATA.SLIDER and creates a new SAS data set called RAWQ that contains data returned from the FABDB2 database. The DBMS connection information was previously assigned to macro variable &FABDB2. The BITESIZE parameter is set to 40 input rows at a time becuase I got an application heap storage size error from the DB2 server when I tried using more.

```
   %DBMSlist(mydata.slider, %str(wafer=#wafer#
   and row=#row#), n n, rawq, &fabdb2, %nrstr(
    select wafer, row, columnn, wafersize as
   size, date, time, stationid as station,
   p145201 as HT, p145202 as EC,P145203,p145205
   from hrs.op1452 where (&mylist) ),
   bitesize=40);
```

SOURCE CODE FOR MACROS

The source code for the three macros %DBMSlist, %MakeList, and %RunQuery are included below. Copy and paste each one to a file by the same name in your autocall macro library to make them visible to your SAS programs. The macro you use is %DBMSlist, the other two are called by this macro. You can copy this code from the CD copy of the Proceedings you receive at the conference, or from the SUGI web site after the conference: http://www.sas.com/usergroups/sugi/proceedings/index.html.

SOURCE CODE FOR MACRO %DBMSLIST

```
%Macro DBMSlist(dsn, column, vtype, newdsn,
    dbname, query, bitesize=200, test=no,
    dlm=%str(#) );

proc sql noprint;
 select count(*) into :gwhxxxx1
 from &dsn;
 quit;

%if &gwhxxxx1=0 %then %do;
 %put ====== WARNING: Input data set &dsn is
    empty, macro ends =======;
 %goto exit;
 %end;

%let totpass=%sysevalf(&gwhxxxx1/&bitesize,
    ceil);

%if &test=no %then %do j=1 %to &gwhxxxx1 %by
    &bitesize;
%let p=%sysevalf(&j/&bitesize, ceil);
%put ================= Starting pass &p of
    &totpass ==================;
data gwhxxxx2;
 set &dsn (firstobs=&j
    obs=%eval(&j+&bitesize-1));
 run;

%MakeList(mylist, gwhxxxx2, &column, &vtype);
%RunQuery(&dbname, gwhxxxx3, &query);

%if &j=1 %then %do;
data &newdsn;
 set gwhxxxx3;
 run;
%end;
%else %do;
Proc append base=&newdsn data=gwhxxxx3;
 run;
%end;
%end;

%else %do; %* Test=yes: do one query for timing;
data gwhxxxx2;
 set &dsn (firstobs=1 obs=&bitesize);
 Run;

%MakeList(mylist, gwhxxxx2, &column, &vtype);
%RunQuery(&dbname, gwhxxxx3, &query);

%end;
%exit: %mend DBMSlist;
```

SOURCE CODE FOR MACRO %MAKELIST

You might need to adjust the formatting for the date, time, and datetime SAS variable types for DBMS systems other than DB2.

```
%macro MakeList(globname, dsn, varinfo,
    vartype);

%local i j;
%global &globname;
%let &globname=; /* return null if macro fails*/

%let numvars=1; %* find number of variables
    specified;
%do %while (%scan(&vartype,%eval(&numvars+1))
    ne);
 %let numvars=%eval(&numvars+1);
 %end;

/**********************************************
 Single variable entered
 **********************************************/
%if &numvars=1 %then %do;

/**************** Character ********/
%if %upcase(&vartype)=C %then %do;
proc sql noprint;
 Select
    Distinct translate(quote(&varinfo),"'",'"')
 into :&globname separated by ','
 from &dsn;
 quit;
%end;

/**************** Numeric ********/
%else %if %upcase(&vartype)=N %then %do;
proc sql noprint;
 select distinct &varinfo
 into :&globname separated by ','
 from &dsn;
 quit;
%end;

/**************** Date ********/
%else %if %upcase(&vartype)=D %then %do;
proc sql noprint;
 select distinct "'"||
    put(&varinfo, yymmdd10.)||"'"
 into :&globname separated by ','
 from &dsn;
 quit;
%end;

/**************** Time ********/
%else %if %upcase(&vartype)=T %then %do;
proc sql noprint;
 select distinct "'"||
    translate(put(&varinfo, time.),
    '.',':','0',' ')||"'"
 into :&globname separated by ','
 from &dsn;
 quit;
%end;

/**************** Datetime ********/
%else %if %upcase(&vartype)=DT %then %do;
proc sql noprint;
 select distinct "'"||put(datepart(&varinfo),
    yymmdd10.)||"-"||
    translate(put(timepart(&varinfo),
    time15.6),'.',':','0',' ')||"'"

 into :&globname separated by ','
 from &dsn;
 quit;
%end;

%else %put ******* Invalid variable type:
    &vartype *************;

%end;  /* %if &numvars=1  */

/**********************************************
 Multiple variables entered
 **********************************************/
%else %do;

/***** Parse SQL template *******************/
%let j=1;
%do %while (%index(%quote(&varinfo), &dlm)>0);
 %let markloc=%index(%quote(&varinfo), &dlm);
 %let text&j=%substr(%quote(&varinfo), 1,
    %eval(&markloc-1));
 %let varinfo=%substr(%quote(&varinfo),
    %eval(&markloc+1),
    %eval(%length(&varinfo)-&markloc) );
 %let markloc=%index(%quote(&varinfo), &dlm);
 %let var&j=%substr(%quote(&varinfo), 1,
    %eval(&markloc-1));
 %if %length(&varinfo)>&markloc %then
    %let varinfo=%substr(%quote(&varinfo),
    %eval(&markloc+1),
    %eval(%length(&varinfo)-&markloc) );
 %else %let varinfo=;
 %let j=%eval(&j+1);
 %end;

/*** Build macro variable with Proc SQL ******/
proc sql noprint;
 select distinct '(' ||

%do i=1 %to &j-1;

/***** Character variable *******************/
%if %upcase(%scan(&vartype, &i))=C %then
    " &&text&i " ||
    translate(quote(&&var&i),"'",'"') ||;

/***** Numeric variable ********************/
%else %if %upcase(%scan(&vartype, &i))=N %then
    " &&text&i " || compress(put(&&var&i,
    best20.)) ||;

/***** Date variable ***********************/
%else %if %upcase(%scan(&vartype, &i))=D %then
    " &&text&i '" || put(&&var&i,
    yymmdd10.)||"'" ||;

/***** Time variable ***********************/
%else %if %upcase(%scan(&vartype, &i))=T %then
    " &&text&i '" ||
    translate(put(&&var&i, time.),'.',':','0',
    ' ')||"'"||;

/***** Datetime variable *******************/
%else %if %upcase(%scan(&vartype, &i))=DT %then
    " &&text&i '" || put(datepart(&&var&i),
    yymmdd10.)||
    "-" ||translate(put(timepart(&&var&i),
    time15.6), '.',':','0',' ')||"'"||;

%else %put ********** Invalid variable type:
    %scan(&vartype,&i) *************;

%end;  /*  %do i=1 %to &j-1  */
```

```
    " &varinfo)" into :&globname separated by
     ' or ' from &dsn;
 quit;
%end;  /* %else for %if &numvars=1  */
%mend MakeList;
```

SOURCE CODE FOR MACRO %RUNQUERY

Change the value for macro variable &DBMStype to the correct value for your DBMS - DB2, Oracle, Teradata, etc.

```
%macro RunQuery(dbinfo, dsname, query);
%let DBMStype=DB2;
proc sql;
 connect to &DBMStype (&dbinfo);
 create table &dsname as select * from
    connection to &DBMStype (
    %unquote(&query) for fetch only);
 %put &sqlxmsg;
 disconnect from &DBMStype;
quit;
%mend RunQuery;
```

CONCLUSION

SAS/ACCESS software provides powerful tools for retrieving data from many heterogenous sources, including most relational database software in use today. Using the techniques described in this paper, you can efficiently join SAS data sets with DBMS tables. If you are using SAS 7 or 8, try the SAS/ACCESS LIBNAME statement with the DBKEY= or DBINDEX= data set options. If you are still using SAS 6.12, try a macro like %DBMSlist presented in this paper.

REFERENCES

SAS Institute Inc. (1999), *SAS/ACCESS Software for Relational Databases: Reference, Version 8,* Cary, NC: SAS Institute Inc.

ACKNOWLEDGEMENTS

Many thanks to Donna Walker at SAS Institute Technical Support for providing unpublished documentation about the DBNULLKEYS= data set option.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Garth W. Helf
IBM Corporation
5600 Cottle Road
San Jose, CA 95193
Work Phone: 408-256-7514
Fax: 408-256-2410
Email: helf@us.ibm.com

TRADEMARK CITATION

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

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