Paper RF07-2015

Macro to Create Multiple SQL IN Clauses from One Column of Data

Mark Millman, Optum, Eden Prairie, Minneapolis, MN

Abstract

When faced with linking data from multiple sources it is often necessary to pull data from one database using a list of IDs from another. While this can be accomplished many ways, using SQL pass-thru is often the best. In order to leverage SQL pass-thru you must supply any needed information from outside the database right in the query. Often you have this information in a dataset outside of the server. This macro separates a given column of data into one or more SQL-compliant "IN" clauses with a count of items that the user specifies.

Introduction

Using SQL pass-thru allows the RDMBS server to do most of the work rather than having the process take place on the machine running SAS® Software. This method is often much faster than declaring the server as a library and referencing it that way. For a fuller discussion see "SAS 9.4 SQL Procedure User's Guide" by SAS Institute.

The main drawback to SQL pass-through is all the SQL must be in the form native to the server and all information needed must be passed-to or contained on the server. A number of solutions are available if you wish to create a list to pass onto a query using SQL pass-through. Here are three examples:

Examples of methods to use a column of foreign data in SQL Pass-Through:

1. Create a Macro Variable

One simple solution is to use PROC SQL to create a macro variable:

```
PROC SQL NOPRINT;
SELECT DISTINCT <column>
INTO :MacroVar SEPARATED BY ","
FROM <dataset>;
QUIT;
```

You can now reference %MacroVar later in your code. This method works great for a limited number of numeric records. What if you have millions of records, or you have character records you wish to surround by a quote? You could create another column of data surrounded by quotes and use the above, however, if you have millions, or perhaps billions of records, most RDBMS will not parse it due to memory limitations.

2. Insert data into a table you have access to on the target server

Another solution is to insert the data into an already created table on the server you wish to query and do a join to it later. Here is an example loading an existing temp (temp_table) table having a column for characters (C1) from Teradata:

```
PROC SQL;
connect to teradata  (server=&mydb. user=&myid. password=&mypwd.
mode=teradata);
INSERT INTO temp_table  (C1)
select distinct
       <CHAR_VALUE>
       FROM <Dataset>;
DISCONNECT FROM teradata;
QUIT;
```

3. **Volatile Temporary Tables**
   Yet another solution is to create a volatile temporary table. If your database allows it, you can create a table only available during the connection. Once you disconnect the table will disappear and no longer be available. Here is an example of creating a volatile temp table on Teradata.

```
PROC SQL;
connect to teradata  (server=&mydb. user=&myid. password=&mypwd.
mode=teradata connection=global);
execute(CREATE VOLATILE TABLE CHAR_LIST
                           (
/*Column Name followed by a space and then the datatype (length if
necessary, integer does not need it) for the column. Use a comma in
between to add more */
                           Charlist varchar(20)
                            ) ON COMMIT PRESERVE ROWS;
              ) by teradata;
QUIT;
```

   Other RDBMS like Oracle and SQL-Server may have different syntax for creating a volatile table. Once your volatile table is created, you can insert rows into it as shown in example 2 and use it in your SQL pass-through joins.

Occasionally you will not have the necessary security to create volatile tables or it is impractical to have your own temporary table on the server. In this case you will need to organize your list of items in your where clause down to a manageable number. This is where the IN clause creating macro comes into play.

Create a sample list to use with the macro

This code creates a simple table with one column of numeric data to demonstrate the macro. It is one column populated with the numbers 1-5.

```
PROC SQL;
   CREATE TABLE work.list
        (
         ID num
        );
INSERT INTO work.list
    values(1)
    values(2)
    values(3)
    values(4)
    values(5);
QUIT;
```

Description of the Macro

The macro collects items into an IN clause of the size of your choosing. The macro also requires you to do something with the clause, in this example, the clauses will be written to a table. This macro can be used anywhere an IN clause is needed. Most often, I have used it to run the same query for each in clause and aggregate the results.

The macro requires the user to supply the following information:
1. The input table having the column of data to be converted to an IN clause
2. The name of the field containing the data
3. Either NONE or SINGLE for no quotes or single quotes around the items in the clause
4. The number of items you want in each clause. The macro currently caps this at 1,000 but that bit of code could easily be removed
5. The name of the output library where the result data should go

Examples of Macro Usage

Let's use the example data to make the macro work. This first example will place all the records into one in clause by choosing 5 as the "chunksize":

Input:

```
%InclauseMaker(table=work.list,field=ID,quote=NONE,chunksize=5,outputlib=
work);
```

Output:

```
In_List
IN(1,2,3,4,5)
```

The next example will place each record into its own clause by choosing a "chunksize" 1:

Input:

```
%InclauseMaker(table=work.list,field=ID,quote=NONE,chunksize=1,outputlib=
work);
```

Output:

```
In_List
IN(1)
IN(2)
IN(3)
IN(4)
IN(5)
```

The next example will use the same concept, but create in clauses with three items instead of 5:

Input:

```
%InclauseMaker(table=work.list,field=ID,quote=NONE,chunksize=3,outputlib=
work);
```

Output:

```
In_List
IN(1,2,3)
IN(4,5)
```

As you can see, with a size of 3, the first 3 records go in the first in clause and the final two go in a subsequent clause.

In the final example I will place quotation marks around the data, which is required for character type data in SQL. In my example the data is numeric, but the macro will still place quotes around it.

Input:

```
%InclauseMaker(table=work.list,field=ID,quote=SINGLE,chunksize=5,outputli
b=work);
```

Output:

In_List

IN('1','2','3','4','5')

Error Checking

The macro checks for the following conditions:

1. The presence of the specified input file
2. That a clause length of 1 or greater is chosen
3. That the specified field existed in the specified table
4. That the input file has at least one record

There are many other errors one could make but these will stop the macro and a message will be put in the log as to which condition was not met.

Conclusion

Using SQL pass-thru is often the most effective way to retrieve data from an RDBMS using SAS. One of the biggest constraints of using SQL pass-thru is supplying data from outside the database into the query. While many solutions to this problem exist, this IN clause creating macro allows you to split a column of data into one or many separate IN clauses that can be used for SQL pass-thru or any other purpose.

Disclaimer

The opinions expressed in this presentation and in the accompanying paper are mine and do not necessarily reflect the opinions of Optum or UnitedHealth Group.

Contact Information

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

Mark Millman
Optum
11000 Optum Circle
Eden Prairie, MN 55344, USA

Work Phone: 952-205-7120
Fax: 877-523-8283
E-mail: mark.millman@optum.com

Full Program Text:

```
*******************************************************************************
********************;
*****  Program name: IN Clause Creator
*****  Description:  Create sets of in clauses based on a field the user
indicates. Length of In clause, use of quotes, and output destination are also
chosen by the user.
*****  Contact:       Mark Millman
*****  Create date:  4/30/13
*****  Rev date:      6/1/15
```

```
*****  Inputs: table containing column of either character or numeric data to
split
*****  Parameters:  Syntax
%InclauseMaker(table=,field=,quote=,chunksize=,outputlib=)

table:
Name of the table with the column you wish to chunk out, do not use quotes
Example:     inlib.macrotest

field: column name in the previously specified table, do not use quotes
Example: ID

quote:
Choose either NONE or SINGLE, do not use quotes
Example: NONE

chunksize: How many observations do want in one chunk do not use quotes. Code
currently limits to 1000 but can be modified
Example: 1000

outputlib:
Libarary where the output will go.
Example: work

*****
*****  Outputs: Creates a string that works as a valid in clause for SQL
statements
*****************************************************************************
********************;

/* good options to turn on for debugging purposes
Options symbolgen mlogic mprint mfile;
*/

*****************************************************************************
****************************************;
/*Create Action Macro to do something with the in clauses created ("&final." is
how to reference them)*/
/*This sample macro writes values to a table. */
%MACRO ActionMacro();
/*Check to see if output table already exists, if not, create it */
%IF %sysfunc(exist(&outputlib..IDCHUNKS)) %THEN %DO;
%END;
%ELSE %DO;
       /*Determine length you need for output table */
       DATA _NULL_;
       CALL SYMPUT('CL',LENGTH("&final."));
       STOP;

       /*Create Output table */
       PROC SQL;
       CREATE TABLE &outputlib..IDCHUNKS
       (
       /*varname type size format     label   */
    In_List    char (&CL.)         label='IN LIST'
       );
       quit;
%END;

/*Write Chunk to table */
PROC SQL;
INSERT INTO &outputlib..IDCHUNKS
values ("&final.");
```

```
QUIT;
%MEND;

********************************************************************************
*************************************************;
/*Main Macro that creates the Chunks */
%MACRO InclauseMaker(table=,field=,quote=,chunksize=,outputlib=);
/*Echo the variables passed to the Macro */
%PUT "Table Chosen: &table";
%PUT "Field Chosen: &field";
%PUT "Quote Chosen: &quote";
%PUT "Chunksize Chosen: &chunksize";
%PUT "Outputlib Chosen: &outputlib";

/* Validate chosen chunk size is 1 or greater */
%IF %EVAL(&chunksize) <1 %THEN %DO;
       %PUT 'Negative or zero chunk sizes are not possible';
       %ABORT;
%END;

/*Safety measure to set chunks >1000 to =1000 */
%IF %EVAL(&chunksize) >1000 %THEN %DO;
       %PUT 'Setting chunksize to 1000 to prevent sql string
                size from exceeding db maximum';
       %LET chunksize = 1000;
%END;


/* Check to make sure ID list table exists */
%IF %sysfunc(exist(&table)) %THEN %DO;
              %PUT 'The input dataset does exist';
%END;
%ELSE %DO;
              %PUT 'The input dataset does not exist';
              %ABORT;
%END;

/* Make sure variable exists in the table */
%LET dsid=%SYSFUNC(OPEN(&table));
%LET check=%SYSFUNC(varnum(&dsid.,&field));
%IF &check = 0 %THEN %DO;
       %PUT 'Variable does not exist';
       %ABORT;
%END;
%ELSE %DO;
       %PUT 'Variable does exist';
%END;

/*Get and store number of table observations */
 %LET dsid=%SYSFUNC(OPEN(&table));
 %LET nobs=%SYSFUNC(ATTRN(&dsid.,NOBS));
 %LET rc=%SYSFUNC(CLOSE(&dsid.));

/*If table has no records, abort the process */
%IF &Nobs. EQ 0 %THEN %DO;
       %PUT 'There are no records in the dataset.';
       %ABORT;
%END;

/*Count how many chunks will be created based on observations and chunksize*/
%PUT "Number of total Chunks: %SYSEVALF(%EVAL(&Nobs)/%EVAL(&chunksize))";
%LET fullchunks=%EVAL(%EVAL(&Nobs)/%EVAL(&chunksize));
%LET remainder=%EVAL(%EVAL(&Nobs)-%EVAL(&fullchunks * &chunksize));
```

```
%PUT "Number of full chunks = &fullchunks";
%PUT "Number of remainder records = &remainder";

/*Create Jay counter Variable */
%Let J=1;
/*Create initial temporary String */
%Let strstore=;

/*Loop through all Records in the Table*/
%DO I=1 %TO &Nobs.;

       /*Set Full chunk to no */
       %Let full = NO;

       /*Reset J to 1 if it has gone to the chunksize */
       %IF &J. > &Chunksize. %THEN %DO;
              %Let J=1;
       %END;

       /* Advance to the Ith record */
       DATA _NULL_;
       SET &table (FIRSTOBS=&I);
       /* store the variables */
       CALL SYMPUT('VAR1',&field);
       STOP;

       /*Enclose value in quotes, if necessary */
       %IF %UPCASE(&quote.) = NONE %THEN %DO;
              DATA _NULL_;
              CALL SYMPUT('temp1',"&VAR1");
              STOP;
       %END;
       %IF %UPCASE(&quote.) = SINGLE %THEN %DO;
              DATA _NULL_;
              CALL SYMPUT('temp1',CATS("'","&VAR1","'"));
              STOP;
       %END;

       /*Add comma if not on last item in chunk */
       %IF &J. < &Chunksize. %THEN %DO;
              DATA _NULL_;
              CALL SYMPUT('temp2',cat("&temp1",","));
              STOP;
       %END;

       /*When chunk size is reached create a comma free temp2 variable */
       %IF &J. = &Chunksize. %THEN %DO;
              DATA _NULL_;
              CALL SYMPUT('temp2',"&temp1");
              STOP;
       %END;

       /*Append temp2 value to storage string */
       DATA _NULL;
       CALL SYMPUT('strstore',CATS("&strstore","&temp2"));
       STOP;

       /* Create the final in clause including parentheses */
       %IF &J. = &Chunksize. %THEN %DO;
              DATA _NULL_;
              CALL SYMPUT('final',CATS("IN(","&strstore.",")"));
              STOP;
              /*Take action with your chunk. This is the first place you must
mention your action macro*/
              %ActionMacro;


              /*reset the string storage variables */
              %LET strstore=;
              /*set full variable to YES */
              %let full = YES;
       %END;

       /*Increment the chunk counter J */
       %LET J = %eval(&J+1);

/*End the I loop */
%END;

/*Output remaining records not comprising a full chunk */
%IF &full = NO %THEN %DO;
       /*Get string store length */
       DATA _NULL_;
              CALL SYMPUT('SL',LENGTH("&strstore."));
              STOP;
       /*Take the final comma away from less than full chunk*/
       DATA _NULL_;
              CALL SYMPUT('strstore2',substr("&strstore.",1,%eval(&SL-1)));
              STOP;

       /* add IN and parentheses around the chunk */
       DATA _NULL_;
       CALL SYMPUT('final',CATS("IN(","&strstore2.",")"));
       STOP;

       /*Take Action with final Chunk. This is the second and final place you
must mention your action macro */
       %ActionMacro;

%END;

%MEND InclauseMaker;
```