Paper CC-02

Joining Wide Tables With PROC SQL

John M. Wildenthal, JPMorgan Chase

ABSTRACT

SQL is easier to use if one does not need to hand code the SELECT clause. Hand coding is particularly onerous for wide tables. This paper shows how to get SAS to do it for you.

MOTIVATION

There are many reasons to use SQL in SAS. The most prominent is that the `DATA/MERGE/BY` construct does not return the Cartesian product as the result set for many-to-many joins. A properly specified SQL join does.

If one is joining datasets `A` and `B` on a common variable `x`, `SELECT A.*, B.*` will fail because SAS sees `x` twice. So to make the query work, either one would need to rename `x` in one dataset, or type in all the field names for one dataset. Having the same variable in twice seems inefficient. Typing all the field names is certainly subject to typos and omissions.

SAS provides the Macro language as a way to dynamically write SAS code. Writing the SELECT clause dynamically would be the best solution.

ELEMENTS

We need to provide our code with lists of the fields in each dataset. We need to compare these lists and delete duplicates from at least one side. We need to enumerate them as part of a `SELECT` clause.

We can get the list of field names at least two ways. `PROC CONTENTS` is probably the best. Using the `DICTIONARY` tables is feasible but often has slower performance. `PROC CONTENTS` can create a dataset with all the information we might ever need – `NAME, TYPE, LENGTH, INFORMAT`.

Comparing these lists is simple. We could `SORT` and use a `DATA/MERGE/BY`, but then we might have to worry about upper/lowercase. Using `PROC SQL` is more natural.

Creating the `SELECT` clause is also most easily done in `PROC SQL`.

CODE:

```
%*****************************************************
**  Get Variable Lists                                **
*****************************************************;

PROC CONTENTS DATA=A OUT=ALIST NOPRINT;
RUN;

PROC CONTENTS DATA=B OUT=BLIST NOPRINT;
RUN;

PROC SQL STIMER NOPRINT;

  %*************************************************
  **  Create the SELECT expressions                **
  *************************************************;

  SELECT UPCASE(NAME)
    INTO :AVARS SEPARATED BY ', A.'
    FROM ALIST;
```

```
  SELECT UPCASE(NAME)
    INTO :BVARS SEPARATED BY ', B.'
    FROM BLIST
    WHERE UPCASE(NAME) NOT IN (SELECT UPCASE(NAME) FROM ALIST);

  %*************************************************
  **  Join the tables                             **
  *************************************************;

  CREATE TABLE JOINED_TABLE AS
  SELECT A.&ALIST, B.&BLIST
    FROM A,
         B,
    WHERE A.X EQ B.X;

QUIT;
```

Now you have a SQL join that is robust to changes in any fields other than the key value field. Fields can come, go, change type, and your query runs without errors and gives you what you expect. Naturally, you can modify this to perform a left, right, or full outer join. If you want variables to overwrite when present, you will want to create a `CASE` expression that determines if the dominant table has a NULL for that join and choose from the appropriate table accordingly. Simply exclude that variable from both of the `SELECT` creation queries through the `WHERE` clause, and hand code that into the join. One could also list them somewhere (macro variable, table) and use that information to exclude them from the simple `SELECT` construction and make them part of another `INTO` clause that creates the `CASE` expressions.

What else can we do? How about saving some space in Oracle? If we put fields in an order that puts NULLs at the end of the row, Oracle will use less space to store the row. I'll assume NULLs across fields are not correlated, that no field name is differs from any other field name by only a leading underscore, and that no field name is at the 32 character limit for v7+ or at the 8 characters for v6 or earlier. For this example, we'll add some `DATA` step processing:

```
%*************************************************
**  Get the fields, their types, etc.            **
*************************************************;

PROC CONTENTS DATA=a OUT=alist NOPRINT;
RUN;

%*************************************************
**  For each field, create the Oracle-specific defn  **
*************************************************;

DATA alist;
  SET alist;
  IF TYPE EQ 2 THEN              %* Character variable ;
    fieldtype = 'VARCHAR2(' || PUT(LENGTH,4.) || ')';
  ELSE                           %* Numeric OR Datetime ;
    IF INDEX(INFORMAT,'DATE')
        OR
        INDEX(INFORMAT,'TIME')
        OR
        INDEX(INFORMAT,'DD')
       THEN
        fieldtype = 'DATETIME';
    ELSE
      fieldtype = 'NUMBER';
RUN;
```

```
%*****************************************************
**  Create statements and clauses to count missings  **
**  for each variable in the table                   **
**                                                   **
**                                                   **
**  Rename every variable to have an underscore      **
**  first                                            **
**                                                   **
**  Count missings into original name                **
**                                                   **
**  Keep original name  [Drop renamed names]         **
*****************************************************;

PROC SQL STIMER NOPRINT;

  SELECT STRIP(name) || '=_' || STRIP(name),
         STRIP(name) || ' + MISSING(_' || STRIP(name) || ')',
         STRIP(name)
    INTO :renaming SEPARATED BY ' ',
         :datacount SEPARATED BY '; ',
         :keeplist SEPARATED BY ' '
    FROM alist;

QUIT;

DATA nmiss1;
  SET a (RENAME=(&renaming)) END=eof;
  &datacount;
  KEEP &keeplist;
  IF eof THEN OUTPUT;
RUN;

%*****************************************************
**  Transpose and rename for ease of use             **
*****************************************************;

PROC TRANSPOSE DATA=nmiss1 OUT=nmiss (RENAME=(_NAME_=NAME col1=NMISS));
RUN;

PROC SQL STIMER NOPRINT;

%*****************************************************
**  Create the clause containing names and defns     **
*****************************************************;

  SELECT STRIP(nmiss.name) || ' ' || STRIP(alist.fieldtype)
    INTO :create_clause SEPARATED BY ', '
    FROM alist,
         nmiss
    WHERE alist.name EQ nmiss.name
    ORDER BY nmiss;

%*****************************************************
**  Connect to Oracle and execute the statement      **
*****************************************************;

  CONNECT TO ORACLE AS ouser (<oracle login>);
```

```
  EXECUTE (CREATE TABLE a (&create_clause)) BY ouser;
QUIT;
```

Now you can use `PROC APPEND` or your preferred method to insert your data.

Other uses of these techniques could allow the program to automatically generate partitioning and subpartitioning clauses; place partitions in specific table spaces; etc.

HELPFUL HINT

When developing code that uses Macro variables, `OPTIONS SYMBOLGEN;` will show you what the variables resolve to at each point they are used. That way you can see if you are generating the code you think you are generating.

CONTACT INFORMATION

Further inquiries are welcome to:

John M. Wildenthal
JPMorgan Chase
Phone: (614) 217-5715 john.m.wildenthal@jpmchase.com