

Flattening Dependent Tables from Relational Databases for Use in SAS Processing

Summary

Increasingly, SAS programmers are tasked with extracting data from relational databases for analyses. One problem encountered by these programmers is that observations stored in two tables with a one-to-many relationship must be accumulated to a single SAS observation prior to analysis. This paper demonstrates and evaluates a number of methods of accomplishing the accumulation and provides a macro based on the most efficient method.

Introduction

By definition, relational databases have a built-in mechanism for linking a record from one table to several records from another table. This mechanism is called a 1-to-many relationship. When analyzing data with SAS, processing is done an observation at a time; therefore, when analyzing data from a relational database, it is necessary to convert the data relationships in the underlying table structure into a single SAS observation. This paper describes an efficient method using Base SAS to accomplish the transformation of a 1-to-many relationship to a single SAS observation, demonstrates it with an example, compares it to other methods, and provides a general SAS macro that performs the transformation.

Fundamentally, the 1-to-many transformation problem is one of reorganizing the data from the table with non-unique records so that it has one observation for each row of the table that has unique records. To facilitate the discussion, let the table with the single row matching many rows be called the parent table and any table that has many rows be denoted as a child table. Also, denote the common field between these tables as the key value.

The reorganizing consists of two steps. First, each child table must be scanned to determine the maximum number of rows in that table that correspond to a key in the parent table. Using that maximum value as a parameter for an array statement in a subsequent data step, the data from the table is read into SAS arrays. The transformation is completed after an observation is created for each key value on the child table. The final step is to merge the transformed data from the child table with the parent table.

Illustrative Example

For example, suppose you have an accounting system that stores sales transactions. The main table Transactions stores the total amount of the sale, the date of the sale, the purchaser, as well as an ID field, which is the key value. Details associated with each transaction are stored in a table named Details. Details contains one row for each type of item purchased and sales tax information. Since many detail lines can be associated with any transaction, the Details table would have a many-to-1 relationship with the table Transactions. Tables 1 and 2 show these tables.

Table 1: *Transactions*, the Parent Database Table

Bill ID	Name	Date	Bill Amount
1	Alice	12-May-06	341.23
2	Betty	22-May-06	417.77

Table 2: *Details*, a Child Database Table

Bill ID	Item	Price	Quantity
1	Item 1	74.99	1
1	Item 3	225.00	1
1	Item 4	15.23	1
1	Sales Tax	26.01	1
2	Item 1	74.99	1
2	Item 2	120.00	1
2	Item 4	15.23	7
2	Item 5	84.33	1
2	Sales Tax	31.84	1

After reading the data into SAS datasets, the first step in merging the data from the child table *Details* with the parent table *Transactions* is determining the maximum number of rows in *Details* that are associated with a particular row in *Transactions*. One way to accomplish this calculation is use a summarizing procedure, like PROC UNIVARIATE, twice—the first pass will determine the number of rows in the child table associated with each key value and the second pass will determine the maximum. The SAS code for this calculation is shown in Figure 1.

Figure 1: Finding the Maximum Key Matches using PROC UNIVARIATE

```
proc sort data=details;
  by bill_id;

proc univariate data=details noprint;
  var bill_id;
  output out=d_count n=count;
  by bill_id;

proc univariate data=d_count noprint;
  var count;
  output out=d_max max=d_max;

data _null_;
  set d_max;
  call symput('gl_max',d_max);
run;
```

The last data step creates a macro variable named `gl_max` that can be used by subsequent SAS steps. Incidentally, the “run;” statement is necessary to force the compilation and execution of the data step, which assigns the macro variable its value.

This maximum value is the number of elements needed in the arrays used to store the data from the child tables. The ‘flattening’ occurs when several observations in the child dataset are reduced to a single observation that can be merged with the parent dataset. For example, in the *Details* table, four rows are associated with Bill ID 1. The flattened data would consist of three SAS arrays (one for item, one for price, and one for quantity), each with four elements along with the key variable `bill_id`. In general, there will

be as many SAS arrays as there are non-key fields in the child table and each array will have as many non-missing elements as the number of rows in the child table associated with its key value.

Figure 2 shows the data step that creates one observation for each key value with arrays for each of the other variables in the Details dataset followed by the merge with the parent dataset. Since numeric values are required in various places in the data step, the %EVAL function is used to resolve the macro variable gl_max as a number. Without this function, the SAS compiler would return an error. The data is read into the data step sorted by the key variable bill_id. Each time a new value of the key variable is encountered, the arrays are initialized. The data step uses a counter to set values of each array element during each pass through the data step. The observation is written when the last occurrence of the key value is processed.

Figure 2: Building SAS Arrays using a DATA Step

```
data flat_detail;
  array xitem{%eval(&gl_max)} $12 xitem1-xitem%eval(&gl_max);
  array xcost{%eval(&gl_max)}      xcost1-xcost%eval(&gl_max);
  array xqty{%eval(&gl_max)}      xqty1- xqty%eval(&gl_max);
  retain counter xitem1-xitem%eval(&gl_max)
             xcost1-xcost%eval(&gl_max)
             xqty1- xqty%eval(&gl_max);

  set details;
  by bill_id;
  if (first.bill_id eq 1) then
    do;
      do i=1 to %eval(&gl_max);
        xitem{i}=.;
        xcost{i}=.;
        xqty{i}=.;
      end;
    end;
  xitem{counter}=item;
  xcost{counter}=cost;
  xqty{counter}=qty;
  if (last.bill_id) then output;
  else delete;

  counter=counter+1;
  keep bill_id xitem1-xitem%eval(&gl_max)
        xcost1-xcost%eval(&gl_max)
        xqty1- xqty%eval(&gl_max);

proc sort data=flat_detail,
  by bill_id;

proc sort data=transactions;
  by bill_id;

data transactions;
  merge flat_detail transactions;
  by bill_id;
```

Table 3 shows the listing of the SAS dataset after the final merge.

Table 3: Dataset Transactions, After Flattening

Obs	bill_id	Name	Date	Bill	Amount
1	1	Alice	12-May-06	341.23	
2	2	Betty	22-May-06	417.77	

Obs	xitem1	xitem2	xitem3	xitem4	xitem5
1	Item 1	Item 3	Item 4	Sales Tax	
2	Item 1	Item 2	Item 4	Item 5	Sales Tax

Obs	xcost1	xcost2	xcost3	xcost4	xcost5
1	74.99	225.00	15.23	26.01	.
2	74.99	120.00	15.23	84.33	31.84

Obs	xqty1	xqty2	xqty3	xqty4	xqty5
1	1	1	1	1	.
2	1	1	7	1	1

Alternative Approaches to the Two Basic Steps

The tasks involved in flattening a child table, (1) finding the dimension of the arrays and (2) reconstructing the child dataset using arrays, can each be done more than one way using SAS. For this paper, each task was coded using two different methods and these methods were called from a macro that took as parameters the calculation methods and the number of observations and variables in the parent and child datasets. The amount of time needed to create the final dataset was measured for a variety of combinations of the inputs to the macro and the results were analyzed in using an Analysis of Covariance technique to determine the most efficient approach.

Two methods were used to find the maximum number of rows from a child table that is associated with any particular key value. The first method used two instances of the UNIVARIATE procedure run one after the other and was illustrated in the example above. The other method used a single pass through a data step to find the needed information. Both methods require that the input data set be sorted by the key value and both need to pass the calculated information to subsequent parts of the program for further processing.

Two methods were used to amalgamate fields from each set of rows with the same key value in a child table into SAS arrays. The first was used in the above example and uses *first.* and *last.* variable processing to read data into SAS arrays. This dataset was merged with the parent dataset to create the final output. The second method employed PROC TRANSPOSE to create arrays that were renamed in individual data steps. The datasets were merged with the parent dataset to create the final dataset.

The Testing Macro and Test Results

The macro used for testing is shown in Appendix 1. It takes as input the number of observations and number of variables in the parent dataset, the number of variables and the maximum number of observations associated with a single key value in the child dataset, the method to determine the maximum rows and the method to amalgamate the

data as well as a seed value to initialize the random number generator. The resources used during each iteration were measured using the FULLSTIMER option in SAS. These measures were read from the SASLOG and used as the responses in an Analysis of Covariance model. The fixed factors in the model were the methods and the dataset size parameters were analyzed as covariates. Both main effect and the interaction of the fixed factors were estimated.

The first step in running the simulation was to construct macro-calling statements. A SAS program was written to generate these statements; this program is shown in Appendix 2. The analysis called the macro 180 times. Each of the maximum finding and array building methods were called ninety times each and each pair of methods was present in one-quarter of the calling statements. The number of observations in the parent dataset was randomly assigned in the range of 50,000 to 200,000. The number of variables in the parent and child datasets was independently and randomly varied between five and twenty-nine and the maximum number of observations in the child dataset associated with any particular key value was randomly varied between seven and twenty. To gather the response data which was the sum of the various times and memory usage for each iteration, a SAS program was written to read the SASLOG file and create the sums and then run the covariance analysis. The analysis was performed using PROC GLM. This program is included in Appendix 3.

Based on the 180 run test, the significant regression factors were number of observations in the parent dataset, the number of variables in the child dataset, and the maximum number of observations associated with any particular key value in the child dataset for all three responses: the user time, that is time spent running SAS code; the system time, that is time spent on system calls like I/O operations; and total memory used or the sum of the memory used by all of the procedures and data steps. There was no interaction detected between the two fixed effects. Least squares means were calculated for both main effects. These means showed that the data step method was significantly quicker in a statistical sense than the dual-UNIVARIATE method based on user time for finding the maximum number of observations in the child dataset associated with a particular key value. System time and memory used were slightly higher for the data step method but neither difference was statistically significant. On all measured responses, the data step method was significantly better than the transpose method to build arrays from data in the child table. Therefore, the most efficient mechanism to flatten a child table for SAS processing is using the data step to both find the maximum number of values and to build the final data set. The results from both the full analysis and the reduced analysis using only the significant factors are given in Appendix 4.

The Final Flattening Macro

Figure 5 is the SAS macro that flattens and merges a child dataset with a parent dataset. It uses data step processing to accomplish both of the major tasks involved in the transformation. Besides creating the output dataset, which can be a permanent SAS dataset, the macro will output the array definitions for later use as an option.

Figure 5: The More Efficient Flattening Macro

```
%macro flatten(parent,child,list,sortvar,outds,file);
/* The flatten macro merges a child dataset that has a many-
 * to-one relationship with that parent dataset. It requires
 * that both datasets have a key in common. The child
 * dataset's observations are converted to SAS arrays before
 * the merge. The parameters used in the macro are:
 *
 *      parent  a SAS dataset with a key variable that has a
 *              unique value for each observation.
 *      child   a SAS dataset with the same key variable that
 *              can contain more than one observation per
 *              key value.
 *      list    a space delimited list of variables from the
 *              child dataset that are to be transformed into
 *              SAS arrays. If this parameter is left blank,
 *              all variables (except for the key variable) in
 *              the child dataset are converted to SAS arrays.
 *      sortvar  is the key variable common to both datasets.
 *      outds   is the output dataset. It can be a permanent
 *              SAS dataset.
 *      file    is a file name for the underlying operating
 *              system where the array definitions are stored
 *              for future use.
 */

%let null=; * This assignment allows logical comparison of
            null strings later in the program;

/* This section sorts the child dataset and reads it to
 * determine the maximum number of observations that are
 * associated with the sort variable.
 */
proc sort data=&child;
    by &sortvar;

data _null_;
    retain _max gl_max;
    set &child end=last;
    by &sortvar;
    if (first.&sortvar eq 1) then _max=0;
    _max=_max+1;
    if (last.&sortvar eq 1 and _max gt gl_max) then gl_max=_max;
    if (last eq 1) then call symput('d_max',gl_max);
run;

/* The datasets procedure is used to extract information
 * about the variables in the child dataset used to build
 * the array statements.
 */
proc datasets nolist;
    contents data=&child out=&child._info noprint;

/* This section retains only the variables that should be
 * transposed (based on the value of the macro variable list)
 * in the information dataset created in the previous step
 */
%if &list NE &null %then
```

```

%do;
  data names;
    %let ii=1;
    %do %while(%scan(&list,&ii,' ') ne &null);
      name="%scan(&list,&ii,' ')"; output;
      %let ii=%eval(&ii+1);
    %end;

  proc sort data=names;
    by name;

  proc sort data=&child._info;
    by name;

  data &child._info;
    merge names(in=i1) &child._info;
    by name;
    if i1;
  %end;

/* This dataset builds three macro variables:
*   list which is the same as previously defined. It is
*   redefined here only because if all variables are
*   to be converted into SAS arrays, it needs to be
*   defined.
*   type is a "vector" (i.e. a space delimited list of
*   character strings) of 1's and 2's that are indicators
*   for numeric and character variables, respectively
*   that are in the same position as the variables in
*   the macro variable list.
*   len is another vector that contains a zero for numeric
*   variables and the length of a character variable.
*/
data _null_;
  length list xtype len $200;
  retain list xtype len ' ';
  set &child._info end=last;
  if (upcase(name) ne "%upcase(&sortvar)") then
    do;
      list=trim(left(list)) || ' ' || trim(left(name));
      if (type eq 1) then
        do;
          xtype=trim(left(xtype)) || ' 1';
          len=trim(left(len)) || ' 0';
        end;
      else do;
          xtype=trim(left(xtype)) || ' 2';
          len=trim(left(len)) || ' ' || trim(left(input(length,best12.)));
        end;
      end;
    if (last eq 1) then
      do;
        call symput('list',trim(left(list)));
        call symput('type',trim(left(xtype)));
        call symput('len', trim(left(len)));
      end;
run;

```

```

/* The child dataset is sorted and read into another dataset
* where SAS arrays are created for each variable listed in
* the macro variable list. FIRST. and LAST. processing is
* used along with a counter to initialize, populate, and
* output the arrays. Also, if the macro variable file is
* not null, the array definitions are written to that file
* for use later.
*
* After the array-creation data step, it is merged with the
* parent dataset to create the output dataset.
*/
proc sort data=&child;
  by &sortvar;

data &outds;
  retain counter
  %let ii=1;
  %do %while(%scan(&list,&ii,' ') ne &>null);
    %do jj=1 %to %eval(&d_max);
      _&ii._&jj
    %end;
    %let ii=%eval(&ii+1);
  %end;
  ;
  %let ii=1;
  %do %while(%scan(&list,&ii,' ') ne &>null);
    %let name=%scan(&list,&ii,' ');
    %let vtype=%scan(&type,&ii,' ');
    %let vlen=%scan(&len ,&ii,' ');
    %if %eval(&vtype) = 1 %then
      %do;
        array _&ii._{%eval(&d_max)}
        %do jj=1 %to %eval(&d_max);
          _&ii._&jj
        %end;
      ;
    %end;
    %else %do;
      array _&ii._{%eval(&d_max)} $%eval(&vlen)
      %do jj=1 %to %eval(&d_max);
        _&ii._&jj
      %end;
    ;
    %end;
    %let ii=%eval(&ii+1);
  %end;
  set &child;
  by &sortvar;
%if &file NE &>null %then
%do;
  if (_n_ eq 1) then
    do;
      file "&file" mod;
    %let ii=1;
    %do %while(%scan(&list,&ii,' ') ne &>null);
      %let name=%scan(&list,&ii,' ');
      vlen=length(trim(left("&name")));
      put @          1 "/* &name" @;
      put + (16-vlen) "*/    array _&ii._{%eval(&d_max)}    " @;
    %end;
  %end;
%end;

```

```

        %do jj=1 %to %eval(&d_max);
            put " _&ii._&jj" @;
        %end;
        put ";";
        %let ii=%eval(&ii+1);
    %end;
end;
%end;
if (first.&sortvar) then
    do;
        counter=1;
        do ii=1 to %eval(&d_max);
            %let ii=1;
            %do %while(%scan(&list,&ii,' ') ne &null);
                _&ii._{ii}=.;
                %let ii=%eval(&ii+1);
            %end;
        end;
    end;
    %let ii=1;
    %do %while(%scan(&list,&ii,' ') ne &null);
        _&ii._{counter}=%scan(&list,&ii,' ');
        %let ii=%eval(&ii+1);
    %end;
    counter=counter+1;
    if (last.&sortvar) then output;
                        else delete;

    keep &sortvar
    %let ii=1;
    %do %while(%scan(&list,&ii,' ') ne &null);
        %do jj=1 %to %eval(&d_max);
            _&ii._&jj
        %end;
        %let ii=%eval(&ii+1);
    %end;
;

proc sort data=&outds;
    by &sortvar;

proc sort data=&parent;
    by &sortvar;

data &outds;
    merge &parent &outds;
    by &sortvar;
%mend flatten;

```

Appendix 1: The SAS Macro Used to Test Different Methods

```
%macro doit(seed,a,b,c,d,e,f);
/* Macro doit takes seven arguments. They are:
 * seed a up to 8-digit number to seed pseudo-random
 * number functions;
 * a the number of observations in the parent dataset;
 * b the number of variables in the parent dataset;
 * c the number of variables in the child dataset;
 * d the maximum number of observations in the child
 * dataset that match with any one key value in
 * the parent dataset;
 * e the method used to determine the maximum; and
 * f the method used to flatten the data.
 */

/* Embed comments in the SASLOG to determine where each
 * section starts.
 */
data _null_;
  put "DCW      Iteration Begin";
  put "DCW      Parameters   &e &f: &a  &b  &c  &d";
run;

/* Create p, a parent dataset, with b variables and a
 * observations, where a and b are macro variables.
 */
data p;
  do match=1 to %eval(&a);
    %do ii=1 %to %eval(&b);
      j%eval(&ii)=ranuni(&seed);
    %end;
    output;
  end;

/* Create c, a child dataset, with the same key values as
 * the parent dataset with a variable number of observations
 * for each key value insuring that at least one of the
 * key values has d observations associated with it. The
 * dataset will have c variables where c and d are macro
 * variables.
 */
data c;
  retain maxval 0;
  do match=1 to %eval(&a);
    loopval=ceil((%eval(&d)*ranuni(&seed)));
    if (loopval lt %eval(&d) and
        maxval lt loopval) then maxval=loopval;
    if ( match eq %eval(&a) and
        maxval lt %eval(&d)) then loopval=%eval(&d);
    do ii=1 to loopval;
      %do ii=1 %to %eval(&c);
        i%eval(&ii)=ranuni(&seed);
      %end;
      output;
    end;
  end;
end;
drop maxval ii loopval;
```

```

/* Sort the child dataset by the key variable and call the
 * macro to find the maximum number of child observations
 * associated with a key value with method e where e is a
 * macro variable and is either:
 *   UNI to indicate the univariate method or
 *   DAT to indicate the data step method.
 * and call the macro to flatten the child data set with
 * method f where f is a macro variable and either
 *   XX to indicate the data step method or
 *   TR to indicate the transpose method.
 */
proc sort data=c;
  by match;

%MOM(c,match,&e);

%flatten(p,c, ,match,&f);

/* Create p_prime, equivalent to the original parent dataset
 * from all, the output dataset from the flatten macro.
 */
data p_prime;
  set all;
keep match
  %do ii=1 %to %eval(&b);
    j%eval(&ii)
  %end;
  ;

/* Create c_prime, equivalent to the original child dataset
 * from all, the output dataset from the flatten macro.
 */
data c_prime;
  %do jj=1 %to %eval(&c);
    array _%eval(&jj)_%eval(&d_max);
  %do ii=1 %to %eval(&d_max);
    _%eval(&jj)_%eval(&ii)
  %end;
  ;
  %end;
  set all;
  j=1;
  do while(j le %eval(&d_max));
    %do ii=1 %to %eval(&c);
      i%eval(&ii)=_%eval(&ii)_{j};
    %end;
    if (_1_{j} ne .) then output;
    j=j+1;
  end;
keep match
  %do ii=1 %to %eval(&c);
    i%eval(&ii)
  %end;
  ;

/* Compare the original parent dataset to the one created
 * from the output of the macro and retain the return code
 * in the macro variable rc1.
proc compare base=p compare=p_prime noprint;

```

```

%let rc1=&sysinfo;

/*  Comparre the original child dataset to the one created
 *   from the output of the macro and retain the return code
 *   in the macro variable rc2.
proc compare base=c compare=c_prime noprint;

%let rc2=&sysinfo;

/*  If both return codes are zero, write that the iteration
 *   was successful to the SASLOG; otherwise, note that the
 *   comparison failed.
 */
data _null_;
  x=&rc1;
  y=&rc2;
  if (x eq 0 and y eq 0) then
    do;
      put "DCW      Iteration Sucessful";
    end;
  else do;
      put "DCW      Iteration Failed";
    end;
run;

/*  Create a trival dataset to retain in the datasets procedure
 *   which implicitly deletes the rest of the datasets in the
 *   WORK library.  This step allows the WORK library to be
 *   cleared for subsequent iterations.
 */
data x;
  x=1;

proc datasets nolist;
  save x;
run;
%mend doit;

%macro MOM(ds,sortvar,engine);
/*  This macro finds the maximums number of observations
 *   in a sorted child dataset by one of two methods.  The
 *   univariate method uses two invocations of PROC UNIVARITE
 *   and the data step method uses a single pass through a
 *   data step to find this maximum value.  The macro variables
 *   passed to this procedure are:
 *
 *       ds the dataset,
 *       sortvar the key variable, and
 *       engine the calculation method
 */
data _null_;
  put "DCW      MOM start";
run;
%global d_max;
%if %upcase(&engine) = UNI %then
  %do;
    proc univariate data=&ds noprint;

```

```

        var &sortvar;
        output out=d_count n=count;
        by &sortvar;

proc univariate data=d_count noprint;
    var count;
    output out=d_max max=gl_max;

    data _null_;
        set d_max;
        call symput('d_max',gl_max);
    run;
%end;
%else %do;
    data _null_;
        retain _max gl_max;
        set &ds end=last;
        by &sortvar;
        if (first.&sortvar eq 1) then _max=0;
        _max=_max+1;
        if (last.&sortvar eq 1 and _max gt gl_max) then gl_max=_max;
        if (last eq 1) then call symput('d_max',gl_max);
    run;
%end;
data _null_;
    put "DCW      MOM end";
run;
%mend MOM;

%macro flatten(parent,child,list,sortvar,engine);
/* This macro builds that flattened child dataset and
 * merges it with the parent dataset by one of two
 * methods. The XX method uses a single pass through
 * a dataset and the TR method used the output of
 * PROC TRANSPOSE to build SAS arrays of the variables
 * in the child dataset passed in the macro variable
 * list.
 *
 * parent the parent dataset,
 * child the child dataset,
 * list a list of variables in the child
 * dataset to convert to SAS arrays.
 * If this variable is blank, all
 * variables in the child dataset will
 * be converted.
 * sortvar the key variable between the parent
 * dataset and child datasets
 * engine the method used to build the arrays
 */

data _null_;
    put "DCW      FLATTEN start";
run;
/* Macro variable initialization
 */
%global glist;
%let null=;
/* The first section of the flatten macro reads the
 * contents of the child dataset into a dataset with

```

```

* the same name as the child dataset appended with _info.
* This data set is used by both method to determine if
* the variables to be converted to SAS arrays are numeric
* or character. The _info dataset is merged with the list
* of variables to retain only those that should be converted.
*/
proc datasets nolist;
    contents data=&child out=&child._info noprint;

%if &list NE &null %then
    %do;
        data names;
            %let ii=1;
            %do %while(%scan(&list,&ii,' ') ne &null);
                name="%scan(&list,&ii,' ')" ; output;
                %let ii=%eval(&ii+1);
            %end;

        proc sort data=names;
            by name;

        proc sort data=&child._info;
            by name;

        data &child._info;
            merge names(in=i1) &child._info;
            by name;
            if i1;
        %end;

/* This is the TRANSPOSE engine
*/
%if %upcase(&engine) = TR %then
    %do;
        /* Build three macro variables for use later.
        * The first, list, is the list of variables to be
        * converted to SAS arrays. The second, leny, is
        * a list of indicator variables that correspond to
        * the variables in the list. If an indicator is
        * zero the variable is numeric; if it is non-zero,
        * the value of the indicator is the length of the
        * character variable in the list. Finally, chflg
        * is a flag to indicate the presence of character
        * variables in the list.
        */
        data _null_;
            length list leny $200;
            retain list leny ' ' charflag 0;
            set &child._info end=last;
            if (upcase(name) ne "%upcase(&sortvar)") then
                do;
                    list=trim(left(list))||' '||trim(left(name));
                    if (type eq 1) then
                        do;
                            leny=trim(left(leny))||' 0';
                        end;
                    else do;
                        leny=trim(left(leny))||' '||
                            trim(left(input(put(length,3.),$3.)));
                    end;
                end;
            if last;
        run;
    %end;

```

```

        charflag=1;
    end;
end;
if (last eq 1) then
do;
    call symput( 'list',trim(left(list)));
    call symput('lenty',trim(left(lenty)));
    call symput('chflg',input(put(charflag,1.),$1.));
end;
run;

/* The child dataset is transposed by the key variable
* and the output is stored in the dataset _temp. For
* each variable in the list a new dataset is created
* that reads the trasposed data for that variable into
* a SAS array coll-colN (where N is the maximum number
* of observations in the child dataset associated with
* a key value) and assigns these values to the SAS
* array _I_1-_I_N (where N was as before and I is the
* number of the variable in the list.
*
* For example, if the list was "dog cat rat" and the
* maximum number of observations in the child dataset
* associated with a key value were 10, then
* (1) the variables dog, cat, and rat would be transposed
*     by the key variable and stored in temp; and
* (2) three datasets would be created, one for each of
*     the variables. The dataset associated with the
*     variable cat would be the second one and would
*     contain the key variable and ten SAS variables
*     _2_1-_2_10.
*
* Also, the array definitons for each variable and the
* variable name is written to a file named test.file for
* use later.
*
* Character and numeric variables are treated differently
* and each temporary dataset is sorted for eventual merging
* with the other temporary datasets.
*/
proc transpose data=&child out=_temp;
    var &list;
    by &sortvar;

%let ii=1;
%do %while(%scan(&list,&ii,' ') ne &null);
%let name=%scan(&list ,&ii,' ');
%let len=%scan(&lenty,&ii,' ');
%if &len = 0 %then %let len=20;
data _xtemp%eval(&ii);
%if %scan(&lenty,&ii,' ') = 0 %then
%do;
    array _&ii._{%eval(&d_max)}
    %do jj=1 %to %eval(&d_max);
        _&ii._&jj
    %end;
;
%if &chflg = 0 %then
%do;

```

```

        array col{%eval(&d_max)} col1-col%eval(&d_max);
        if (_n_ eq 1) then
            do;
                vlen=length(trim(left("&name")));
                file "test.file" mod;
                put @          1 "/* &name" @;
                put + (16-vlen) "*/   array _&ii._{%eval(&d_max)}      " @;
                %do jj=1 %to %eval(&d_max);
                    put " _&ii._&jj" @;
                %end;
                put ";";
            end;
        %end;
    %else %do;
        array col{%eval(&d_max)} %eval(&len) col1-col%eval(&d_max);
        if (_n_ eq 1) then
            do;
                vlen=length(trim(left("&name")));
                file "test.file" mod;
                put @          1 "/* &name" @;
                put + (16-vlen) "*/   array _&ii._{%eval(&d_max)}      " @;
                %do jj=1 %to %eval(&d_max);
                    put " _&ii._&jj" @;
                %end;
                put ";";
            end;
        %end;
    %end;
%else %do;
    array _&ii._{%eval(&d_max)} %eval(&len)
    %do jj=1 %to %eval(&d_max);
        _&ii._&jj
    %end;
;
    array col{%eval(&d_max)} %eval(&len) col1-col%eval(&d_max);
    if (_n_ eq 1) then
        do;
            vlen=length(trim(left("&name")));
            file "test.file" mod;
            put @          1 "/* &name" @;
            put + (16-vlen) "*/   array _&ii._{%eval(&d_max)} %eval(&len)" @;
            %do jj=1 %to %eval(&d_max);
                put " _&ii._&jj" @;
            %end;
            put ";";
        end;
    %end;
    set _temp;
    if _name_ eq "&name";
    %do jj=1 %to %eval(&d_max);
        %if %scan(&lenty,&ii,' ') = 0 %then
            %do;
                _&ii._{%&jj}=col{%&jj};
            %end;
        %else %do;
            _&ii._{%&jj}=input(put(col{%&jj},%eval(&len).),best%eval(&len).);
        %end;
    %end;
keep &sortvar

```

```

        %do jj=1 %to %eval(&d_max);
            _&ii._&jj
        %end;
    ;

proc sort data=_xtemp%eval(&ii);
    by &sortvar;

%let ii=%eval(&ii+1);
%end;

/* This section merges the temporary datasets together
 * to form the final outupt dataset.
 */
data all;
    merge &parent
        %do ii=%eval(&ii-1) %to 1 %by -1;
            _xtemp%eval(&ii)
        %end;
    ;
    by &sortvar;

/* Delete temporary datasets from the WORK library.
 */
proc datasets nolist;
    delete
        %let ii=1;
        %do %while(%scan(&list,&ii,' ') ne &null);
            _xtemp%eval(&ii)
            %let ii=%eval(&ii+1);
        %end;
    ;
run;
%end;
/* This is the DATA STEP engine
 */
%else %do;
    /* Build three macro variables for use later.
     * The first, list, is the list of variables to be
     * converted to SAS arrayys. The second, xtype, is
     * a list of indicator variables that correspond to
     * the variables in the list. If an indicator is
     * one the variable is numeric; if it is two the
     * variable is character. The thrid variable, len,
     * is similar to the second and contains a zero for
     * numeric variables and the length of a charater
     * variable.
     *
     * NOTE: A similar technique as used by the TRANSPOSE
     * engine above would have reduced the number
     * of macro variables by one.
     */
data _null_;
    length list xtype len $200;
    retain list xtype len ' ';
    set &child._info end=last;
    if (upcase(name) ne "%upcase(&sortvar)") then
        do;
            list=trim(left(list)) || ' ' || trim(left(name));

```

```

        if (type eq 1) then
            do;
                xtype=trim(left(xtype))||' 1';
                len=trim(left(len)) ||' 0';
            end;
            else do;
                xtype=trim(left(xtype))||' 2';
                len=trim(left(len)) ||' '||trim(left(input(length,best12.)));
            end;
        end;
    if (last eq 1) then
        do;
            call symput('list',trim(left(list )));
            call symput('type',trim(left(xtype)));
            call symput('len', trim(left(len )));
        end;
run;

/* The data from the child dataset is read directly into
* SAS arrays. When the value of the sort variable changes,
* the arrays are initialized and the arrays are output
* when just before the value of the sort variable changes
* using "first." and "last." processing.
*
* The flattened child dataset is merged with the parent
* dataset to create all, the output dataset.
*
* As in the TRANSPOSE engine, the array definitions for
* each variable and the variable name is written to a file
* named test.file for use later.
*/
proc sort data=&child;
    by &sortvar;

data all;
    retain counter
    %let ii=1;
    %do %while(%scan(&list,&ii,' ') ne &null);
        %do jj=1 %to %eval(&d_max);
            _&ii._&jj
        %end;
        %let ii=%eval(&ii+1);
    %end;
;
    %let ii=1;
    %do %while(%scan(&list,&ii,' ') ne &null);
        %let name=%scan(&list,&ii,' ');
        %let vtype=%scan(&type,&ii,' ');
        %let vlen=%scan(&len ,&ii,' ');
        %if %eval(&vtype) = 1 %then
            %do;
                array _&ii._{%eval(&d_max)}
                %do jj=1 %to %eval(&d_max);
                    _&ii._&jj
                %end;
            ;
        %end;
        %else %do;
            array _&ii._{%eval(&d_max)} $%eval(&vlen)

```

```

                %do jj=1 %to %eval(&d_max);
                    _&ii._&jj
                %end;
            ;
        %end;
        %let ii=%eval(&ii+1);
    %end;
    set &child;
    by &sortvar;
    if (first.&sortvar) then
        do;
            counter=1;
            do ii=1 to %eval(&d_max);
                %let ii=1;
                %do %while(%scan(&list,&ii,' ') ne &null);
                    _&ii._{ii}=.;
                    %let ii=%eval(&ii+1);
                %end;
            end;
        end;
        %let ii=1;
        %do %while(%scan(&list,&ii,' ') ne &null);
            _&ii._{counter}=%scan(&list,&ii,' ');
            %let ii=%eval(&ii+1);
        %end;
        counter=counter+1;
        if (last.&sortvar) then output;
                                else delete;

        keep &sortvar
        %let ii=1;
        %do %while(%scan(&list,&ii,' ') ne &null);
            %do jj=1 %to %eval(&d_max);
                _&ii._&jj
            %end;
            %let ii=%eval(&ii+1);
        %end;
    ;

    proc sort data=all;
        by &sortvar;

    proc sort data=&parent;
        by &sortvar;

    data all;
        merge &parent all;
        by &sortvar;

    %end;
/* The list of variables in the array statements
 * is passed to the global macro variable glist.
 */
%let glist="&list";
data _null_;
    put "DCW      FLATTEN end";
run;
%mend flatten;

```

Appendix 2: Macro-call Building Program

```
data one;
/* This program generates macro calls of the form:
 *   %doit(seed,a,b,c,d,e,f);
 * to repeated call the test macro.
 */
seed=331448139;
do i=1 to 180;
    if (i le 90 and mod(i,2) eq 0) then endx=',UNI,XX);';
    else if (i le 90 and mod(i,2) eq 1) then endx=',UNI,TR);';
    else if (i gt 90 and mod(i,2) eq 0) then endx=',DAT,XX);';
    else if (i gt 90 and mod(i,2) eq 1) then endx=',DAT,TR);';
    random=input(put(floor(ranuni(seed)*10000000),7.),$7.);
    iter=0;
    do while (iter lt 50000 or iter gt 200000);
        iter=ranuni(seed)*200100;
    end;
    iterx=input(put(floor(iter),6.),$6.);
    pvar=0;
    do while (pvar lt 5 or pvar gt 30);
        pvar=ranuni(seed)*31;
    end;
    pvarx=input(put(floor(pvar),2.),$2.);
    cvar=0;
    do while (cvar lt 5 or cvar gt 30);
        cvar=ranuni(seed)*31;
    end;
    cvarx=input(put(floor(cvar),2.),$2.);
    maxc=0;
    do while (maxc lt 7 or maxc gt 21);
        maxc=ranuni(seed)*22;
    end;
    maxcx=input(put(floor(maxc),2.),$2.);
    string='%doit('||right(random)||','||
           ||right(iterx)||','||
           ||right(pvarx)||','||
           ||right(cvarx)||','||
           ||right(maxcx)||endx;
    file 'macro.calls';
    put string;
end;
```

Appendix 3: SAS Program used to Extract Data from SASLOG and Perform Analysis

```
options nodate nonumber;
title  'Full Factorial Model with All Covariates';
title2 'Data from 180 Esecutions of the Test Macro';
filename data 'sim3_180.log';

data one;
/* This data step reads in the log (beginning after the listing
 * of the program) and searches for lines beginning with DCW
 * to find the beginning and ending of various steps. It also
 * searches for timing and memory usage statistics embedded
 * in the SASLOG. Here are some lines from the SASLOG that
 * was analyzed showing examples of the types of lines that
 * are read and processed by this program:
 *
 * 581          %doit(5521840, 93539, 5, 8, 7,UNI,TR);
 *
 * DCW      Iteration Begin
 * DCW      Parameters      UNI TR: 93539  5  8  7
 * NOTE: DATA statement used (Total process time):
 *      real time          0.01 seconds
 *      user cpu time      0.01 seconds
 *      system cpu time    0.00 seconds
 *      Memory                                135k
 *
 *
 * NOTE: The data set WORK.P has 93539 observations and 6 variables.
 * NOTE: DATA statement used (Total process time):
 *      real time          0.19 seconds
 *      user cpu time      0.12 seconds
 *      system cpu time    0.01 seconds
 *      Memory                                152k
 */
retain max_meth arr_meth key_vals par_vars ch_vars ch_mxky
       rsum      usum      ssum      msum
       rtsum     utsum     stsum     mtsum
       m_real    m_user    m_sys     m_mem
       f_real    f_user    f_sys     f_mem;
infile data firstobs=618;
input;
if (_infile_ ne '') then
  do;
    if (substr(_infile_,1,3) eq 'DCW') then
      do;
        select (scan(_infile_,2,' '));
          when ('MOM')
            do;
              if (scan(_infile_,3,' ') eq 'start') then
                do;
                  rsum=0;
                  usum=0;
                  ssum=0;
                  msum=0;
                end;
            else do;

```

```

        m_real=rsum;
        m_user=usum;
        m_sys=ssum;
        m_mem=msum;
    end;
end;
when ('FLATTEN')
do;
    if (scan(_infile_,3,' ') eq 'start') then
    do;
        rsum=0;
        usum=0;
        ssum=0;
        msum=0;
    end;
    else do;
        f_real=rsum;
        f_user=usum;
        f_sys=ssum;
        f_mem=msum;
    end;
end;
when ('Parameters')
do;
    max_meth=scan(_infile_,3,' ');
    arr_meth=scan(_infile_,4,' :');
    key_vals=input(put(scan(_infile_,5,' '),$3.),3.);
    par_vars=input(put(scan(_infile_,6,' '),$3.),3.);
    ch_vars=input(put(scan(_infile_,7,' '),$3.),3.);
    ch_mxky=input(put(scan(_infile_,8,' '),$3.),3.);
end;
when ('Iteration')
do;
    if (scan(_infile_,3,' ') eq 'Sucessful') then
    do;
        t_real=rtsum;
        t_user=utsum;
        t_sys=stsum;
        t_mem=mtsum;
        output;
    end;
    else if (scan(_infile_,3,' ') eq 'Begin') then
    do;
        rtsum=0;
        utsum=0;
        stsum=0;
        mtsum=0;
    end;
    else
    do;
        file print;
        put "ERRORs encountered during processiong";
    end;
end;
otherwise
do;
    file print;
    put "Unexpected number of DCW comments";
end;

```

```

        end;
        delete;
    end;
    if (length(trim(left(_infile_))) ge 16) then
        testval=substr(trim(left(_infile_)),1,16);
        else delete;
    if (testval in
        ('real time' 'user cpu time' 'system cpu time' 'Memory')) then
    do;
        if (testval eq 'Memory') then
            do;
                str=put(scan(_infile_,2,' '),$7.);
                if (substr(str,length(str),1) eq 'b') then mult=1;
                if (substr(str,length(str),1) eq 'k') then mult=1024;
                if (substr(str,length(str),1) eq 'm') then mult=1024*1024;
                strip=substr(str,1,length(str)-1);
                msum=msum +(input(put(strip,$7.),7.)*mult);
                mtsum=mtsum+(input(put(strip,$7.),7.)*mult);
            end;
        else if (substr(_infile_,length(_infile_)-6,7) eq 'seconds') then
            do;
                if (testval eq 'real time') then
                    do;
                        rsum=rsum +input(put(scan(_infile_,3,' '),$7.),7.2);
                        rtsum=rtsum+input(put(scan(_infile_,3,' '),$7.),7.2);
                    end;
                else if (testval eq 'user cpu time') then
                    do;
                        usum=usum +input(put(scan(_infile_,4,' '),$7.),7.2);
                        utsum=utsum+input(put(scan(_infile_,4,' '),$7.),7.2);
                    end;
                else if (testval eq 'system cpu time') then
                    do;
                        ssum=ssum +input(put(scan(_infile_,4,' '),$7.),7.2);
                        stsum=stsum+input(put(scan(_infile_,4,' '),$7.),7.2);
                    end;
            end;
        end;
    else do;
        if (testval eq 'real time') then
            do;
                rsum=rsum +input(put(scan(_infile_,3,' '),$12.),stimer12.);
                rtsum=rtsum+input(put(scan(_infile_,3,' '),$12.),stimer12.);
            end;
        else if (testval eq 'user cpu time') then
            do;
                usum=usum +input(put(scan(_infile_,4,' '),$12.),stimer12.);
                utsum=utsum+input(put(scan(_infile_,4,' '),$12.),stimer12.);
            end;
        else if (testval eq 'system cpu time') then
            do;
                ssum=ssum +input(put(scan(_infile_,4,' '),$12.),stimer12.);
                stsum=stsum+input(put(scan(_infile_,4,' '),$12.),stimer12.);
            end;
        end;
    end;
    end;
    delete;
end;
keep max_meth arr_meth key_vals par_vars ch_vars ch_mxky
    m_real    m_user    m_sys    m_mem

```

```

        f_real    f_user    f_sys    f_mem
        t_real    t_user    t_sys    t_mem;

proc glm data=one;
  class max_meth arr_meth;
  model t_user t_sys t_mem=
    max_meth arr_meth key_vals par_vars ch_vars ch_mxky
    max_meth*arr_meth
    max_meth*key_vals max_meth*par_vars max_meth*ch_vars max_meth*ch_mxky
    arr_meth*key_vals arr_meth*par_vars arr_meth*ch_vars arr_meth*ch_mxky/
    ss3;
run;

title  'Main-Effects Model with Significant Covariates';
title2 'Data from 180 Esecutions of the Test Macro';
proc glm data=one;
  class max_meth arr_meth;
  model t_user t_sys t_mem =
    max_meth arr_meth key_vals ch_vars ch_mxky
    max_meth*key_vals max_meth*ch_vars max_meth*ch_mxky
    arr_meth*key_vals arr_meth*ch_vars arr_meth*ch_mxky/ss3;
  lsmeans max_meth arr_meth/pdiff adjust=dunnett;
run;

```

Appendix 4: SAS Listing of Analysis of Covariance Comparing the Different Methods

Full Factorial Model with All Covariates
Data from 180 Esecutions of the Test Macro

The GLM Procedure

Class Level Information

Class	Levels	Values
max_meth	2	DAT UNI
arr_meth	2	TR XX

Number of Observations Read	180
Number of Observations Used	180

Full Factorial Model with All Covariates
Data from 180 Esecutions of the Test Macro

The GLM Procedure

Dependent Variable: t_user

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	15	137241.8657	9149.4577	43.88	<.0001
Error	164	34192.1595	208.4888		
Corrected Total	179	171434.0252			

R-Square	Coeff Var	Root MSE	t_user Mean
0.800552	42.17497	14.43914	34.23628

Source	DF	Type III SS	Mean Square	F Value	Pr > F
max_meth	1	676.11871	676.11871	3.24	0.0736
arr_meth	1	15.26668	15.26668	0.07	0.7870
key_vals	1	13133.86407	13133.86407	63.00	<.0001
par_vars	1	99.36143	99.36143	0.48	0.4910
ch_vars	1	44851.41112	44851.41112	215.13	<.0001
ch_mxky	1	6672.77528	6672.77528	32.01	<.0001
max_meth*arr_meth	1	0.19200	0.19200	0.00	0.9758
key_vals*max_meth	1	40.11870	40.11870	0.19	0.6615
par_vars*max_meth	1	60.71574	60.71574	0.29	0.5902
ch_vars*max_meth	1	240.71536	240.71536	1.15	0.2842
ch_mxky*max_meth	1	245.38664	245.38664	1.18	0.2796
key_vals*arr_meth	1	2345.31726	2345.31726	11.25	0.0010
par_vars*arr_meth	1	39.43881	39.43881	0.19	0.6642
ch_vars*arr_meth	1	16880.20022	16880.20022	80.96	<.0001
ch_mxky*arr_meth	1	35.27489	35.27489	0.17	0.6814

Full Factorial Model with All Covariates
Data from 180 Esecutions of the Test Macro

The GLM Procedure

Dependent Variable: t_sys

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	15	28909.56074	1927.30405	39.64	<.0001
Error	164	7973.85552	48.62107		
Corrected Total	179	36883.41626			

R-Square	Coeff Var	Root MSE	t_sys Mean
0.783809	35.45493	6.972881	19.66689

Source	DF	Type III SS	Mean Square	F Value	Pr > F
max_meth	1	2.21659	2.21659	0.05	0.8312
arr_meth	1	212.91993	212.91993	4.38	0.0379
key_vals	1	3891.06621	3891.06621	80.03	<.0001
par_vars	1	74.52270	74.52270	1.53	0.2175
ch_vars	1	15732.30225	15732.30225	323.57	<.0001
ch_mxky	1	5105.56893	5105.56893	105.01	<.0001
max_meth*arr_meth	1	28.00195	28.00195	0.58	0.4490
key_vals*max_meth	1	74.06709	74.06709	1.52	0.2189
par_vars*max_meth	1	1.77457	1.77457	0.04	0.8487
ch_vars*max_meth	1	4.93483	4.93483	0.10	0.7504
ch_mxky*max_meth	1	28.10449	28.10449	0.58	0.4482
key_vals*arr_meth	1	14.84984	14.84984	0.31	0.5813
par_vars*arr_meth	1	47.23280	47.23280	0.97	0.3258
ch_vars*arr_meth	1	198.81831	198.81831	4.09	0.0448
ch_mxky*arr_meth	1	335.86004	335.86004	6.91	0.0094

Full Factorial Model with All Covariates
Data from 180 Esecutions of the Test Macro

The GLM Procedure

Dependent Variable: t_mem

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	15	4.7721122E18	3.1814081E17	65.53	<.0001
Error	164	7.961825E17	4.8547713E15		
Corrected Total	179	5.5682947E18			

R-Square	Coeff Var	Root MSE	t_mem Mean
0.857015	26.88455	69676189	259168142

Source	DF	Type III SS	Mean Square	F Value	Pr > F
max_meth	1	6.9536539E15	6.9536539E15	1.43	0.2331
arr_meth	1	5.0124365E16	5.0124365E16	10.32	0.0016
key_vals	1	2.7504584E17	2.7504584E17	56.65	<.0001
par_vars	1	9.1426047E14	9.1426047E14	0.19	0.6649
ch_vars	1	8.0854396E17	8.0854396E17	166.55	<.0001
ch_mxky	1	1.6616959E17	1.6616959E17	34.23	<.0001
max_meth*arr_meth	1	4.8821007E14	4.8821007E14	0.10	0.7516
key_vals*max_meth	1	3.2262778E15	3.2262778E15	0.66	0.4161
par_vars*max_meth	1	2.136292E15	2.136292E15	0.44	0.5080
ch_vars*max_meth	1	7.2147714E15	7.2147714E15	1.49	0.2246
ch_mxky*max_meth	1	2.8207838E15	2.8207838E15	0.58	0.4470
key_vals*arr_meth	1	1.5658037E17	1.5658037E17	32.25	<.0001
par_vars*arr_meth	1	3.5874789E15	3.5874789E15	0.74	0.3912
ch_vars*arr_meth	1	7.5005448E17	7.5005448E17	154.50	<.0001
ch_mxky*arr_meth	1	1.1086047E17	1.1086047E17	22.84	<.0001

Main-Effects Model with Significant Covariates
Data from 180 Esecutions of the Test Macro

The GLM Procedure

Class Level Information

Class	Levels	Values	
max_meth	2	DAT UNI	
arr_meth	2	TR XX	
Number of Observations Read			180
Number of Observations Used			180

Main-Effects Model with Significant Covariates
Data from 180 Esecutions of the Test Macro

The GLM Procedure

Dependent Variable: t_user

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	11	137034.0443	12457.6404	60.84	<.0001
Error	168	34399.9810	204.7618		
Corrected Total	179	171434.0252			

R-Square	Coeff Var	Root MSE	t_user Mean
0.799340	41.79631	14.30950	34.23628

Source	DF	Type III SS	Mean Square	F Value	Pr > F
max_meth	1	658.84280	658.84280	3.22	0.0746
arr_meth	1	82.30888	82.30888	0.40	0.5269
key_vals	1	13371.16696	13371.16696	65.30	<.0001
ch_vars	1	45771.54754	45771.54754	223.54	<.0001
ch_mxky	1	6714.28607	6714.28607	32.79	<.0001
key_vals*max_meth	1	43.72616	43.72616	0.21	0.6446
ch_vars*max_meth	1	234.16703	234.16703	1.14	0.2864
ch_mxky*max_meth	1	249.56779	249.56779	1.22	0.2712
key_vals*arr_meth	1	2477.76301	2477.76301	12.10	0.0006
ch_vars*arr_meth	1	17390.96645	17390.96645	84.93	<.0001
ch_mxky*arr_meth	1	32.52336	32.52336	0.16	0.6907

Main-Effects Model with Significant Covariates
Data from 180 Esecutions of the Test Macro

The GLM Procedure

Dependent Variable: t_sys

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	11	28740.38072	2612.76188	53.90	<.0001
Error	168	8143.03554	48.47045		
Corrected Total	179	36883.41626			

R-Square	Coeff Var	Root MSE	t_sys Mean
0.779222	35.39997	6.962072	19.66689

Source	DF	Type III SS	Mean Square	F Value	Pr > F
max_meth	1	7.72101	7.72101	0.16	0.6903
arr_meth	1	159.47852	159.47852	3.29	0.0715
key_vals	1	3950.46283	3950.46283	81.50	<.0001
ch_vars	1	16066.17140	16066.17140	331.46	<.0001
ch_mxky	1	5199.33286	5199.33286	107.27	<.0001
key_vals*max_meth	1	64.74241	64.74241	1.34	0.2494
ch_vars*max_meth	1	7.24044	7.24044	0.15	0.6996
ch_mxky*max_meth	1	27.74312	27.74312	0.57	0.4504
key_vals*arr_meth	1	25.63517	25.63517	0.53	0.4681
ch_vars*arr_meth	1	236.36519	236.36519	4.88	0.0286
ch_mxky*arr_meth	1	328.34318	328.34318	6.77	0.0101

Main-Effects Model with Significant Covariates
Data from 180 Esecutions of the Test Macro

The GLM Procedure

Dependent Variable: t_mem

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	11	4.7648975E18	4.331725E17	90.58	<.0001
Error	168	8.0339722E17	4.7821263E15		
Corrected Total	179	5.5682947E18			

R-Square	Coeff Var	Root MSE	t_mem Mean
0.855719	26.68265	69152920	259168142

Source	DF	Type III SS	Mean Square	F Value	Pr > F
max_meth	1	4.9705382E15	4.9705382E15	1.04	0.3094
arr_meth	1	9.1748764E16	9.1748764E16	19.19	<.0001
key_vals	1	2.7415961E17	2.7415961E17	57.33	<.0001
ch_vars	1	8.2877423E17	8.2877423E17	173.31	<.0001
ch_mxky	1	1.6601917E17	1.6601917E17	34.72	<.0001
key_vals*max_meth	1	3.6707232E15	3.6707232E15	0.77	0.3822
ch_vars*max_meth	1	6.8497688E15	6.8497688E15	1.43	0.2331
ch_mxky*max_meth	1	3.2372804E15	3.2372804E15	0.68	0.4118
key_vals*arr_meth	1	1.6308226E17	1.6308226E17	34.10	<.0001
ch_vars*arr_meth	1	7.7178776E17	7.7178776E17	161.39	<.0001
ch_mxky*arr_meth	1	1.1074141E17	1.1074141E17	23.16	<.0001

Main-Effects Model with Significant Covariates
Data from 180 Esecutions of the Test Macro

The GLM Procedure
Least Squares Means
Adjustment for Multiple Comparisons: Dunnett-Hsu

max_meth	t_user LSMEAN	H0:LSMean1= LSMean2 Pr > t
DAT	31.9502459	0.0460
UNI	36.3438631	

max_meth	t_sys LSMEAN	H0:LSMean1= LSMean2 Pr > t
DAT	19.8166354	0.8170
UNI	19.5701606	

max_meth	t_mem LSMEAN	H0:LSMean1= LSMean2 Pr > t
DAT	256071310	0.9663
UNI	255624310	

Main-Effects Model with Significant Covariates
Data from 180 Esecutions of the Test Macro

The GLM Procedure
Least Squares Means
Adjustment for Multiple Comparisons: Dunnett-Hsu

arr_meth	t_user LSMEAN	H0:LSMean1= LSMean2 Pr > t
TR	48.0237036	<.0001
XX	20.2704054	

arr_meth	t_sys LSMEAN	H0:LSMean1= LSMean2 Pr > t
TR	21.4903518	0.0008
XX	17.8964442	

arr_meth	t_mem LSMEAN	H0:LSMean1= LSMean2 Pr > t
TR	354807421	<.0001
XX	156888199	