**PhUSE 2011**

**Paper CS03**

# Many to many, too many performance tests

Christoph Baumer, Biometrical Practice BIOP, Basel, Switzerland

**Abstract**
There are many known tricks for speeding up the execution of our programs. However due to the complexities of SAS we often get surprising results. Performance testing is often the only way to figure out the fastest way of solving a problem. I will introduce a modular and flexible approach of performance testing. Some programs and algorithms perform well on a small number of observations, but slow down considerably when the number of observations gets higher, for example in megatrials. The performance testing program handles different scenarios, such as running all programs in different orders, running programs in a single batch job or running the program multiple times. The testing program will consolidate the test results in a single dataset, which provides a method to decide a suitable approach before the actual data arrives. An example of different "many to many" joins is provided to illustrate the performance testing procedure.

## Introduction

SAS provides numerous methods for Data Manipulation. Most of us follow an approach which is convenient and not necessarily the most efficient. At times when comparisons are made between different methods of programming it is not always done in a very transparent way. In my paper I want to present a way which takes a closer look to the method of comparing different solutions for a certain problem. This idea leads to a transparent and flexible approach for maintenance testing.

The program was written for MS Windows systems, because MSDOS commands such as DIR are used. Comparing different Operating systems or processors is out of scope of the approach, but not many changes would be required to do so.

The three different scenarios that will be considered are:
1. Running a single program in a batch job.
2. Running a program multiple times.
3. Running programs in different orders.

The above mentioned scenarios are my suggestions for adding variety into the tests that might have very different outcomes.
Scenario number 1 is the most intuitive case one would expect because we often see effectiveness of a program separated from the context. However, we all know that SAS is very good at adding its own optimization measures to the program execution. These

measures are not exactly clear, but I have often observed that say for example the first query to a dictionary dataset can take some time and a second similar query executes much faster.
Scenario number 2 looks less like a typical case when you test for speed, but in our day-to-day work, we often come across the situation where we are re-running the same, slightly changed program over and over again and many of us would be happy if the program creating our lab dataset would only take 10% of its time at the second run!
Scenario number 3 seems to be very close to the real world situation and if we could increase the speed by changing the order, it would be an easy yet surprising approach.

**Example Programs**
To demonstrate the features of the maintenance testing program I will show several approaches of a many to many merge when merging together ATC terms with drug codes using the WHO dictionary.

The 3 datasets used for this merging are THG, ATC and MP.

The dataset ATC contains the different **A**natomic **T**herapeutic **C**lasses (defined by the WHO), which provide classifications of different effect types. A certain drug can be classified in many different ATC classes due to the variety of its potential impact.
The dataset MP contains all drug names defined by the WHO such as a numeric drug code and drug name.
THG contains the merge information and it shows all pairs of drugs with its related ATC code.

```
create table WHO.ATC( label='WHO ATC codes')
  (
   ATCcode char(10) format=$10. label='ATC code',
   ATClevel char(1) format=$1. label='ATC level',
   ATCtxt char(100) format=$120. label='ATC text'
  );


create table WHO.MP( label='WHO medical product')
  (
   medprod num format=10. label='Medicinalprod_Id',
   drecno char(6) label='Drug record number',
   seq1 char(2) label='Sequence number 1',
   seq2 char(3) label='Sequence number 2',
   drugname char(80) label='Drug name',
   namespec char(30) label='Name specifier',
   country char(10) label='Country',
   maholder char(10) label='MA holder',
   create char(8) label='Create date',
   change char(8) label='Change date',
   drg_spec char(110) label='Drug name !! Name specifier',
   drugcode char(8)
  );
```

```
create table WHO.THG( label='WHO therapeutic group' )
  (
   medprod num format=10. label='Medicinalprod_Id',
   ATCcode char(10) format=$10. label='ATC code',
   official char(1) format=$1. label='official'
  );
```

The merged dataset I want to create has the drug name (variable DRUGNAME) together with the ATC Text (variable ATCTXT). It follows the dataset THG but it has the actual names of the drug and of the ATC term instead the (alpha-) numeric codes.

I wrote the following programs to arrive at the solution:

Program 1:

```
%let description=sql inner join;
proc sql;
  create table atc1 as
    select atc.atctxt, mp.drugname
      from thg
        inner join atc on thg.atccode eq atc.atccode
        inner join mp on thg.medprod eq mp.medprod;
quit;
```

Program 2:

```
%let description=sql join with where clause;
proc sql;
  create table atc2 as
    select atc.atctxt, mp.drugname
      from thg, atc, mp
      where thg.atccode eq atc.atccode and thg.medprod eq mp.medprod;
quit;
```

Program 3:

```
%let description=point loops with mp inside;
data atc3;
  set thg;
  do k = 1 to nobs_atc;
    set atc(rename = (atccode = _atccode_)) nobs=nobs_atc point=k;
    if atccode eq _atccode_ then do;
      do l = 1 to nobs_mp;
        set mp(rename = (medprod = _medprod_)) nobs=nobs_mp point=l;
        if medprod = _medprod_ then
          output;
      end;
    end;
  end;
  keep atctxt drugname;
run;
```

Program 4:

```
%let description=point loops with atc inside;
data at4;
  set thg;
  do k = 1 to nobs_mp;
    set mp(rename = (medprod = _medprod_)) nobs=nobs_mp point=k;
    if medprod = _medprod_ then do;
      do l = 1 to nobs_mp;
        set atc(rename = (atccode = _atccode_)) nobs=nobs_atc point=l;
        if atccode eq _atccode_ then
          output;
      end;
    end;
  end;
  keep atctxt drugname;
run;
```

Program 5:

```
%let description=Using formats;
proc sql;
  create table fmt_mp as
  select medprod as start
       , drugname as label
       , 'mp' as fmtname, 'n' as type
         from mp;
  create table fmt_atc as
  select atccode as start
       , atctxt as label
       , 'atc' as fmtname
       , 'c' as type
         from atc;
quit;
proc format cntlin=fmt_mp;
run;
proc format cntlin=fmt_atc;
run;
data atc5;
  set thg;
  acttxt = put(atccode,atc.);
  drugname = put(medprod,mp.);
  keep atctxt drugname;
run;
```

Program 6:

```
%let description=sorting and merging;
proc sort data=atc;
    by atccode;
run;
proc sort data=thg;
    by atccode;
run;
data atc0;
    merge atc thg;
    by atccode;
    keep atctxt medprod;
run;
```

```
proc sort data=mp;
    by medprod;
run;
proc sort data=atc0;
    by medprod;
run;
data atc0;
    merge atc0 mp;
    by medprod;
    keep atctxt drugname;
run;
```

All of these programs are stored in separate files named as prog1, prog2 etc... Each program contains a macro variable named “description” which is read by the maintenance program to add descriptive information about the method chosen. This description together with the other results of the maintenance testing will be stored later in a consolidated results dataset.

**Base programs**

In addition to the executed programs, I also defined different base programs which add information about the requirements of the actual situation. The “base” programs are intended to be the basis of the “prog” files and can be used to create different scenarios such as choosing datasets to be used by “prog” with different numbers of observations or specifying libnames with different paths.
The following base files were used in the example:

Base 1:

```
libname who "D:\many_to_many\lib";
data mp;
    set who.mp;
run;
data thg;
    set who.thg;
run;
data atc;
    set who.atc;
run;
```

Base 2:

```
libname who "D:\many_to_many\lib";
data thg;
    set who.thg;
    where official eq 'N';
run;
proc sql;
    create table mp as
        select * from who.mp where medprod in
            (select medprod from thg);
    create table atc as
        select * from who.atc where atccode in
            (select atccode from thg);
quit;
```

Base 3:

```
libname who "D:\many_to_many\lib";
data thg;
    set who.thg;
    if _n_ le 2000;
run;
proc sql;
    create table mp as
        select * from who.mp where medprod in
            (select medprod from thg);
    create table atc as
        select * from who.atc where atccode in
            (select atccode from thg);
quit;
```

Base 1 copies the three datasets THG, MP and ATC to the work library, Base 2 copies the datasets to the work as well but only takes a subset based on the flag variable *official* and Base 3 does the same, but the subset is based on the variable _n_.

These different conditions result in a different number of observations in the outcome dataset:

| Base condition | Number of observations in result dataset |
|---|---|
| BASE1 | 1826677 |
| BASE2 | 494285 |
| BASE3 | 2000 |

**Maintenance Program**

The main functionality of the maintenance program is to write SAS programs representing different scenarios. These programs will then be executed as batch jobs. To create these SAS Programs, a folder has to be specified in which all programs can be stored. The maintenance program then reads in all files that start with the term “prog” and all files names that start with “base” and creates one consolidated file for each possible combination.

If the files in the main folder are as follows:

BASE1.SAS
BASE2.SAS
PROG1.SAS
PROG2.SAS
PROG3.SAS

Then all possible combinations will be created and 6 files are created and all of these combinations will be run as a batch job:

BASE1.SAS + PROG1.SAS
BASE2.SAS + PROG1.SAS
BASE1.SAS + PROG2.SAS
BASE2.SAS + PROG2.SAS
BASE1.SAS + PROG3.SAS
BASE2.SAS + PROG3.SAS

In addition, there are 3 macro variables that add control to the maintenance testing:

**&repeats:** Specifies the number of repeats of a single program within a single run. This will add a program multiple times to a certain file / batch job.

**&n_runs:** Specifies how many times each batch job is executed.

**&multiple:** Specifies, if two programs are run within the same batch job. For each program, all other programs will be added to the single batch job Starting with the list above, this will extend the list significantly:
BASE1.SAS + PROG1.SAS + PROG2.SAS
BASE1.SAS + PROG1.SAS + PROG3.SAS
…
BASE1.SAS + PROG2.SAS + PROG1.SAS
BASE1.SAS + PROG2.SAS + PROG3.SAS
…

The consolidated program that is created includes a time measurement of the execution and writes the result into a dataset RESULTS.

A snippet of the pseudo code for these programs, e. g. for the combination
"BASE1.SAS + PROG1.SAS + PROG2.SAS"
is as follows:

```
%inc(BASE1)
<store start time>
%inc (PROG1.SAS)
<calculate execution time>
<store results in the dataset RESULTS>
<store start time>
%inc (PROG2.SAS)
<calculate execution time>
<store results in the dataset RESULTS>
```

**Results**

To start comparing the 6 different programs I started by using only BASE3, which has the datasets THG, ATC and MP with a relatively small number of observations (THG has 2000 observations only).

To be quite sure there is no bias, I set the number of runs to 10:

```
%perf(n_runs=10);
```

The result of the different runs is as follows:

| Compare programs with BASE3 as basis | | |
|---|---|---|
| Description | Filename | Mean duration (seconds) |
| sql inner join | prog1.sas | **0.0171** |
| sql join with where clause | prog2.sas | 0.0203 |
| point loops with mp inside | prog3.sas | 1.7721 |
| point loops with atc inside | prog4.sas | 2.0143 |
| Using formats | prog5.sas | 0.0782 |
| Sorting and merging | prog6.sas | 0.0515 |

It is clear from these results that the point loops are not that effective. We can also see that it is advantageous to have the bigger dataset in the inner loop. In this scenario, the inner SQL join appears to be the quickest solution.

Following these results, I removed the programs PROG3.SAS and PROG4.SAS from the main folder and they will not be considered for the remaining maintenance tests.

Now let's have a look at the influence of running a single program multiple times in the same batch job. For this task, I added the two base files BASE1.SAS (the full WHO dictionary) and BASE2.SAS (a subset excluding all official terms) with 5 runs and 3 repeats:

```
%perf(n_runs=5,repeats=3);
```

| Compare programs with BASE1, running each program 3 times | | | |
|---|---|---|---|
| | Mean duration (seconds) | | |
| Description | First repeat | Second repeat | Third repeat |
| sql inner join | 53.7402 | 112.0068 | 101.496 |
| sql join with where clause | 67.5392 | 123.6426 | 170.0498 |
| Using formats | **50.8864** | 139.6062 | **89.4168** |
| Sorting and merging | 230.2746 | **101.7114** | 95.9858 |

One can see that unlike in the first example, now the usage of formats is the most effective program for the first run. In the second run, the mean execution time for most programs increases, but for the sorting and merging approach, the second run is much quicker.

| Compare programs with BASE2, running each program 3 times | | | |
|---|---|---|---|
| | Mean duration (seconds) | | |
| Description | First repeat | Second repeat | Third repeat |
| sql with inner join | **12.4676** | **9.9488** | **13.3372** |
| sql join with where clause | 13.6166 | 20.8902 | 26.3024 |
| Using formats | 18.4868 | 24.088 | 20.0874 |
| Sorting and merging | 45.0796 | 27.1772 | 27.1866 |

When looking at these results, where BASE2 contains datasets with lower numbers of observations than BASE1, not only the time decreases for the sorting and merging as before, but also for the SQL inner join. The SQL inner join is under all conditions the fastest programs in this situation.

The last results I want to show are based on running different programs within the same batch job. This will be done 10 times:

```
%perf(n_runs=10,multiple=YES);
```

This call means, that each combination of two different programs will be executed 10 times. For this run, BASE2 was used.

| **Compare programs with BASE2, running all combinations of programs** | | | | | |
|---|---|---|---|---|---|
| | Mean duration in seconds | | | | |
| Previous / Description | sql inner join | sql join with where clause | Using formats | Sorting and merging | Program run at first place |
| sql inner join | **NA** | 13.6367 | 11.4738 | 8.3364 | 11.3405 |
| sql join with where clause | 12.9232 | **NA** | 12.7544 | 9.5263 | 10.9484 |
| Using formats | 19.799 | 20.4817 | **NA** | 17.0711 | 17.2770 |
| Sorting and merging | 51.2156 | 51.7964 | 50.1929 | **NA** | 44.8809 |

This table shows the mean duration of the different programs (in rows) depending on which other programs ran before (names in columns). The last column shows the mean execution time when the program was run at first place. For this scenario, it seems that only the sorting and merging program improves the execution time of the following program. For all other combinations no huge influence can be seen, but the tendency is that the previous run slightly increases the execution time. It is interesting to see, that unlike in the previous tables, where the time was around two fold for the second run when using BASE1, now the influence of a different previous program is not very strong.

**Resume**

The provided approach is meant to be a starting point for finding out the most effective approach for a certain situation. A very simplified example was provided, but the maintenance program can be used for real life situations as well. All that is needed, is to copy the programs you want to compare into the main folder and specify the condition under which the programs are running by adding different BASE programs.

Even from the simple examples, you can see that the results are not always what you might expect and the best approach is highly dependant on the circumstances. However, a simulation tool like this one can help you prepare your programs before the final data arrives to choose your preferred programs and to avoid unexpected intolerable performance issues when timelines are becoming tight.

## References

none


## Acknowledgements

Many thanks to Rohit Banga and David Garbutt for the review of this paper.


## Contact Information

Comments and questions are welcome and appreciated. Please contact:


Christoph Baumer
Biometrical Practice BIOP
Centralbahnstrasse 9
4051 Basel
Switzerland
E-mail: christoph.baumer@biop.ch

## Appendix

Program code for the main macro:

```
%macro perf(n_runs=1,repeats=1,multiple=NO);
options mprint noxwait;
%let projpath=D:\many_to_many\progs;
%let saspath=C:\Program Files\SAS 9.2\SASFoundation\9.2\sas.exe;

libname rec "&projpath";
filename rec "&projpath";
filename dir pipe "dir ""&projpath"" /b";

%macro create_bat(options=,nr=);
filename run_bat "&projpath.\run&nr..bat";
data _null_;
    file run_bat;
    put " ""&saspath"" -sysin ""&projpath.\run.sas"" &options ";
stop;
run;
%mend create_bat;
%create_bat(options=,nr=1);

/* In previous versions, also the usage of different batch options was forseen, but it is
not used any longer */
/*
%create_bat(options=%str(-nolog),nr=2);
%create_bat(options=%str(-memlib),nr=3);
%create_bat(options=%str(-memlib -nolog),nr=4);
*/

%let set=0;
data prog base run;
    length filename $20;
    infile dir truncover;
    input filename 1-50;
    if index(lowcase(filename),'base') then output base;
    else if index(lowcase(filename),'prog') then output prog;
    else if index(lowcase(filename),'run') and index(lowcase(filename),'.bat') then
output run;
    else if index(lowcase(filename),'records.sas7bdat') then
        call symput('set','1');
run;

%if &set eq 0 %then %do;
proc sql;
    create table rec.records(
        description varchar(200),
        duration numeric,
        basename varchar(20),
        progname varchar(20),
        repeat numeric,
        starttime varchar(20),
        endtime varchar(20),
        order varchar(100),
        batchfile varchar(20)
        );
quit;
%end;

proc sql noprint;
    create table combinations as
    select a.filename as base, b.filename as prog, c.filename as batchfile
        from base as a inner join prog as b on
            not missing(a.filename) and not missing(b.filename)
            inner join run as c on not missing(c.filename);
%let n_comb = &sqlobs;
quit;

%if &multiple ne NO %then %do;
data combinations;
```

```
    set combinations;
    do k = 1 to nobs;
        set prog point=k nobs=nobs;
        if filename ne prog then do;
            prog2 = filename;
            output;
        end;
    end;
run;
proc sql noprint;
    select count(*) into :n_comb from combinations;
quit;
%end;

%start:
%if &n_runs eq 0 %then %do;
%goto end_of_mac;
%end;

filename run "&projpath.\run.sas";
%do i = 1 %to &n_comb;

data _null_;
    length order $100;
    k = &i ;
    set combinations point=k;
    file run;

    put '%let projpath=D:\many_to_many\progs;';
    put 'libname rec "&projpath";';
    put 'filename rec "&projpath";';
    put ' ';
    put '%let description=no description;';
    put ' ';
    put '%inc rec(' base ');';
    put ' ';
    %if &multiple eq NO %then %do;
    do i = 1 to &repeats;
        order = '';
    %end;
    %else %do;
    do i = 1 to 2;
        order = strip(prog) || ', ' || prog2;
        if i = 2 then do;
            prog = prog2;
        end;
    %end;
        put 'data _null_;';
        put '  x = sleep(2);';
        put 'run;';
        put ' ';
        put 'data _null_;';
        put '    datetime =
input(put(date(),date9.)||":"||put(time(),time13.4),datetime22.3);';
        put '    datetime_c = put(datetime,datetime20.3);';
        put '    call symput("start",datetime_c);';
        put 'run;';
        put ' ';
        put '%inc rec(' prog ');';
        put ' ';
        put 'data _null_;';
        put '    datetime =
input(put(date(),date9.)||":"||put(time(),time13.4),datetime22.3);';
        put '    datetime_c = put(datetime,datetime20.3);';
        put '    call symput("end",datetime_c);';
        put 'run;';
        put ' ';
        put 'data new_rec;';
        put '    length basename progname batchfile $20 description $200. order $100. ;';
        put '    basename = "' base '";';
        put '    progname = "' prog '";';
```

```
        put '    repeat = ' i ';';
        put '   starttime = "&start";';
        put '   endtime = "&end";';
        put '    duration = round(input("&end",datetime22.3)-
input("&start",datetime22.3),0.001);';
        put '    description  = "&description";';
        put '    batchfile  = "' batchfile '";';
        put '    order  = "' order '";';
        put 'run;';
        put ' ';
        put 'proc append base=rec.records new=new_rec; ';
        put 'run;';
        put ' ';
    end;
    exec = "x  ' ""&projpath.\"||batchfile||" "";';   ";
    call execute(exec);
    x = sleep(1);
stop;
run;

%end;

%let n_runs=%eval(&n_runs-1);
%goto start;
%end_of_mac:
%mend perf;
```