PharmaSUG China 2025 - Paper CC-136

Identifying New and Updated Records: SAS PROC COMPARE vs. SAS Hashing

Keting Lv, Daiichi Sankyo (China) Holdings Co., Ltd.

ABSTRACT

In clinical research, particularly in data management, efficient tracking of changes across evolving datasets is critical for ensuring robust data review. This paper evaluates two SAS methodologies for identifying new and modified records: **PROC COMPARE**, a native SAS procedure, and **hashing**, an in-memory processing technique.

PROC COMPARE merges historical and current datasets, systematically identifying additions and variable-level updates through automated outputs, making it ideal for structured data. SAS Hashing dynamically stores prior records in memory, enabling rapid, key-based comparisons without requiring sorted datasets, which optimizes performance for large or unsorted data.

While PROC COMPARE offers simplicity and built-in reporting, its scalability is limited for high-volume data. SAS Hashing excels in speed and flexibility but requires advanced coding. Recommendations suggest using PROC COMPARE for routine audits and hashing for scalable, real-time updates. Aligning the method with dataset size, technical resources, and reporting needs enhances data review efficiency, ensuring stakeholders focus on actionable insights.

INTRODUCTION

Clinical trial data management necessitates continuous monitoring of dataset changes to ensure accuracy and regulatory compliance. As datasets evolve (e.g., weekly/monthly listings updates), identifying new or modified records becomes crucial for efficient review. Traditional subject listings - though valuable - become unwieldy as data volume grows, obscuring recent changes. This paper compares two SAS techniques for change tracking:

- **PROC COMPARE**: A structured, procedure-driven approach.
- **SAS Hashing**: A dynamic, in-memory method using DATA step hash objects.

METHODS

PROC COMPARE APPROACH

This method identifies new/updated records using sequential dataset merging and comparison:

Key Requirements:

- Historical (oldcm) and current (newcm) datasets.
- Unique key variables (e.g., SUBJECT, CMTRT, CMSTDTC, CMENDTC).
- Sorted datasets (mandatory for merging).

Example Input Data:

SUBJECT	CMTRT	CMDOSFRM	CMROUTE	CMSTDTC	CMENDTC
1001	PEGINTRON	INJECTION	SUBCUTANEOUS	2004-01-01	2004-01-01
1002	RIBAVIRIN	TABLET	ORAL	2004-01-02	2004-01-02
1002	BOCEPREVIR	TABLET	ORAL	2004-01-03	2004-01-03
1003	PEGINTRON	TABLET	ORAL	2004-01-07	2004-01-07

Display 1. Historical Data (oldcm)

SUBJECT	CMTRT	CMDOSFRM	CMROUTE	CMSTDTC	CMENDTC

1001	PEGINTRON	INJECTION	SUBCUTANEOUS	2004-01-01	2004-01-01
1002	RIBAVIRIN	INJECTION	SUBCUTANEOUS	2004-01-02	2004-01-02
1002	BOCEPREVIR	TABLET	ORAL	2004-01-03	2004-01-03
1003	PEGINTRON	TABLET	ORAL	2004-01-07	2004-01-07
1003	RIBAVIRIN	TABLET	ORAL	2004-01-07	2004-01-07

Display 2. Current Data (newcm)

Steps:

1.Identify New Records:

```
data new_records check(keep=subject cmtrt cmstdtc cmendtc);
  merge oldcm (in=a) newcm (in=b);
  by subject cmtrt cmstdtc cmendtc;
  if b and not a then output new_records;
run;
```

Output:

SUBJECT	CMTRT	CMDOSFRM	CMROUTE	CMSTDTC	CMENDTC
1003	RIBAVIRIN	TABLET	ORAL	2004-01-07	2004-01-07

Display 3. New Records (new_records)

2.Extract Common Records for Comparison:

```
data oldcm2;
  merge check (in=a) oldcm;
  by subject cmtrt cmstdtc cmendtc;
  if a;
run;

data newcm2;
  merge check (in=a) newcm;
  by subject cmtrt cmstdtc cmendtc;
  if a;
run;
```

3.Flag Updated Variables via PROC COMPARE:

```
proc compare noprint base=oldcm2 compare=newcm2 outnoequal out=compout;
  id subject cmtrt cmstdtc cmendtc;
run;
```

Outputs a dataset (compout) with X markers denoting changed variables.

Obs	_TYPE_	_OBS_	SUBJECT	CMTRT	CMSTDTC	CMENDTC	CMDOSFRM	CMROUTE
1	DIF	3	1002	RIBAVIRIN	2004-01-02	2004-01-02	XXXXX.XXX	XXXXXXXXXXXX

Display 4. Output from the PROC COMPARE

4.Generate Human-Readable Change Flags:

```
data compout2;
  length datflag $200;
  set compout;
  array vars [*] $ cmdosfrm cmroute; /* List non-key variables */
  datflag = "";
  do i=1 to dim(vars);
    if index(vars[i], 'X') then
      datflag = catx(', ', datflag, upcase(vname(vars[i])));
  end;
  keep subject cmtrt cmstdtc cmendtc datflag;
run;
```

Obs	datflag	SUBJECT	CMTRT	CMSTDTC	CMENDTC
1	CMDOSFRM, CMROUTE	1002	RIBAVIRIN	2004-01-02	2004-01-02

Display 5. Dataset with Change Descriptions

5.Merge Results for Final Output:

```
data final;
  merge new_records (in=a) newcm compout2;
  by subject cmtrt cmstdtc cmendtc;
  if a then datflag = 'NEW'; /* Flag new records */
run;
```

Output: A dataset with datflag indicating new records (N) or updated variables (e.g., CMDOSFRM, CMROUTE).

Obs	SUBJECT	CMTRT	CMDOSFRM	CMROUTE	CMSTDTC	CMENDTC	datflag
1	1001	PEGINTRON	INJECTION	SUBCUTANEOUS	2004-01-01	2004-01-01	
2	1002	BOCEPREVIR	TABLET	ORAL	2004-01-03	2004-01-03	
3	1002	RIBAVIRIN	INJECTION	SUBCUTANEOUS	2004-01-02	2004-01-02	CMDOSFRM, CMROUTE
4	1003	PEGINTRON	TABLET	ORAL	2004-01-07	2004-01-07	
5	1003	RIBAVIRIN	TABLET	ORAL	2004-01-07	2004-01-07	NEW

Display 6. Final output with datflag for listing review

SAS HASHING APPROACH

Leverages in-memory hash objects for unsorted data and high-volume scalability:

Steps:

1.Load Historical Data into Hash Object:

```
data final2;
  length h_cmdosfrm $50 h_cmroute $50 datflag $200;
  if _n_ = 1 then do;
    declare hash h(dataset: "oldcm(rename=(cmdosfrm=h_cmdosfrm
cmroute=h_cmroute");
    h.defineKey('subject', 'cmtrt', 'cmstdtc', 'cmendtc');
    h.defineData('h_cmdosfrm','h_cmroute');
    h.defineDone();
  end;
```

```
    set newcm;
    /* Check for new/updated records */
    if h.check() ne 0 then datflag = 'NEW';
    end;
    else do;
       datflag = ''; /* Initialize */
      /* Compare non-key variables */
      if cmdosfrm ne h_cmdosfrm then
        datflag = catx(', ', datflag, 'CMDOSFRM');
      if cmroute ne h_cmroute then
        datflag = catx(', ', datflag, 'CMDOSFRM');
    end;
    keep subject cmtrt cmdosfrm cmroute cmstdtc cmendtc datflag;
  run;
```

Advantages:

- No sorting required.
- Single pass processing ideal for large datasets.
- Real-time updates.

COMPARATIVE ANALYSIS

Criterion	PROC COMPARE	SAS Hashing
Ease of Use	Low-code; built-in outputs.	Requires advanced DATA step programming.
Scalability	Limited by merge/sort operations.	High (in-memory processing).
Sorting	Mandatory (adds pre-processing overhead).	Not required.
Reporting	Automated difference reports (OUTNOEQUAL).	Manual flagging.
Performance	Slower for large datasets.	Faster for unsorted/big data.

Table 1. Method Comparison

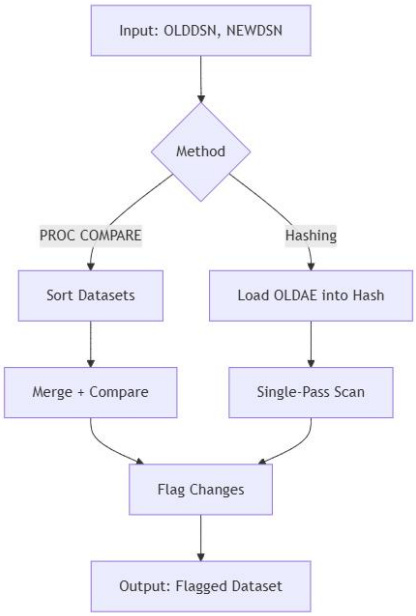

Figure 1. Change Flagging Workflow

Comparative Insight: PROC COMPARE is optimal for small-scale audits due to its simplicity and automated reporting. SAS hashing, while coding-intensive, outperforms in scalability and speed for unsorted/large datasets (>1M records). The choice hinges on data volume, technical expertise, and real-time needs—PROC COMPARE suits standardized environments, while hashing benefits dynamic pipelines.

RECOMMENDATIONS

PROC COMPARE:

- Routine audits of small/medium datasets.
- Teams needing minimal coding and built-in reports.
- Structured data with consistent sorting keys.

SAS HASHING:

- Real-time ETL pipelines or large datasets (>1M records).
- Unsorted data or dynamic keys.
- Advanced programmers optimizing for speed.

CONCLUSION

Both methods efficiently track dataset changes but cater to distinct scenarios. PROC COMPARE simplifies change detection with automated reporting, while SAS Hashing offers unmatched scalability for complex or high-volume data. Aligning the method with organizational needs—dataset size, technical expertise, and reporting frequency—ensures sustainable data management. Future work could explore hybrid approaches or cloud-based optimizations.

REFERENCES

[1] SAS Institute Inc. (2016). SAS 9.4 DATA Step Syntax Reference. Cary, NC: SAS Institute.

ACKNOWLEDGMENTS

The author would like to thank Daiichi Sankyo Biostatistics and Data Management Department for reviewing the paper and providing valuable feedback.

CONTACT INFORMATION

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

Keting Lv
Daiichi Sankyo (China) Holdings Co., Ltd.
86-18106850068
lv.keting.kn@daiichisankyo.com.cn
https://bit.ly/Portfolio-klyu

APPENDIX I: DEMO CODE OF SAS HASHING COMPARE

```
%macro compare_datasets(olddsn=, newdsn=, keys=, outdsn=);
   /* Step 1: Get list of non-key variables */
   proc contents data=&newdsn out=varlist noprint;
   run;

   /* Get dataset name without library */
   %let dsname = %scan(&newdsn, -1, '.');

   proc sql noprint;
      select name into :nonkeys separated by ' '
      from varlist
      where upcase(name) not in (%upcase(%sysfunc(prxchange(s/\s+/','/,-1,"&keys"))))
        and memname = "%upcase(&dsname)";
   quit;

   /* Step 2: Process datasets with error handling */
   data &outdsn;
      length datflag $200;
      drop rc i;

      /* Declare temporary variables for historical values */
      %let n_vars = %sysfunc(countw(&nonkeys));
      %do i = 1 %to &n_vars;
         %let var = %scan(&nonkeys, &i);
         retain h_&var;  /* Add h_ variables to PDV */
      %end;

      if _n_ = 1 then do;
         /* Create rename string for non-key variables */
         %let rename_list =;
         %do i = 1 %to &n_vars;
            %let var = %scan(&nonkeys, &i);
            %let rename_list = &rename_list &var=h_&var;
         %end;

         /* Load historical data with renamed variables */
         declare hash h(dataset: "&olddsn(rename=(&rename_list))");

         /* Define keys - convert space-delimited list to comma-delimited */
         %let keylist = %sysfunc(tranwrd(&keys, %str( ), %str(,)));
         h.defineKey(&keylist);

         /* Define data - all renamed non-key variables */
         %let datalist =;
         %do i = 1 %to &n_vars;
            %let var = %scan(&nonkeys, &i);
            %let datalist = &datalist h_&var;
            %if &i < &n_vars %then %let datalist = &datalist,;
         %end;
         h.defineData(&datalist);
         h.defineDone();
      end;

      set &newdsn;
```

```
        /* Initialize change flag */
        datflag = '';

        /* Check record status */
        rc = h.find();
        if rc ne 0 then do;
           datflag = 'NEW';  /* New record */
        end;
        else do;
           /* Compare all non-key variables */
           %do i = 1 %to &n_vars;
              %let var = %scan(&nonkeys, &i);
              if &var ne h_&var then
                 datflag = catx(', ', datflag, "%upcase(&var)");
           %end;

           /* Set flag if no changes found */
           if datflag = '' then datflag = 'NO CHANGE';
        end;

        /* Keep original variables + datflag */
        keep &keys &nonkeys datflag;
     run;

     /* Cleanup temporary datasets */
     proc datasets lib=work nolist;
        delete varlist;
     run;
     quit;
%mend compare_datasets;

%compare_datasets(
     olddsn = [input historical dataset],
     newdsn = [input current dataset],
     keys   = [space-separated key variables],
     outdsn = [output dataset]
);
```