Paper 991-2016

Using SAS® software to shrink the data in your applications

Ahmed Al-Attar, AnA Data Warehousing Consulting LLC, McLean, VA

ABSTRACT

This paper discusses the techniques I used at the Census Bureau to overcome the issue of dealing with large amounts of data while modernizing some of their public-facing web applications by using service oriented architecture (SOA) to deploy JavaScript web applications powered by SAS®. The paper covers techniques that resulted in reducing **1,753,926** records (**82 MB**) down to **58** records (**328 KB**), a **99.6%** size reduction in summarized data on the server side.

INTRODUCTION

Over the past few years, the Dissemination Internet Staff (DIS) team at the Census Bureau have been working hard to modernize and enhance some of their existing public facing web applications. They chose JavaScript as a Front-End technology of choice, because it can

- Integrate with SAS/IntrNet®
- Get embedded in HTML page
- Operate without Java Application Server
- Enables responsive web design

Based on the above, a custom JavaScript framework was built to provide the required Rich Interactivity, and Integration with SAS® 9.2 through Web Services standard protocols such as REST.

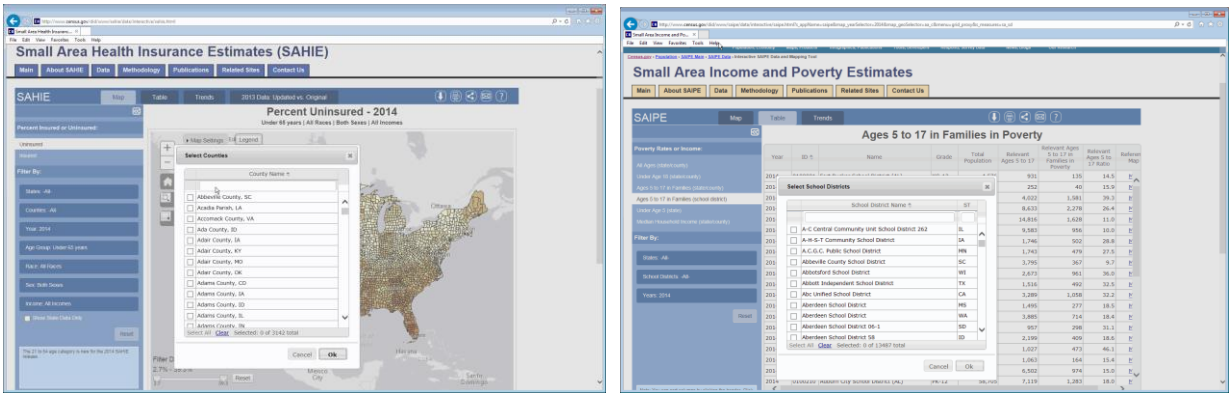


Figure 1. Sample Census Public facing Web applications

PROBLEM/ISSUE

While the client application (JavaScript) can communicate with SAS via submitting HTTP request, the response (data) always returned as json stream. That is where we started to have issues when dealing with large data!

1. WEB APPLICATION DATA SIZE

Many deployed JavaScript web applications

- Follow the same layout which comprised of two parts
  - Data Filters Part: Two or more data selectors widgets with data driven JSON lookup data
  - Data Viewer Part: Data visualization widget, such as Data Grids, Charts and Maps
- Perform server side data selection validations to avoid Zero result sets returned.
  - This proved troublesome and required some attention and thinking outside of the box!

I have always used PROC SUMMARY/PROC MEANS to figure out the unique combinations of variables values. But when the final number of unique combinations exceeds the hundreds and starts to range in the thousands, tens of thousands, and hundreds of thousands, it starts to fail to load during the client application initialization, which in turn causes the application to crash!!

Here are few examples of the variable combinations I had to deal with

Variables Set	# Unique Combinations	Disk Usage	Notes
Year, State/County	70,239	640 **KB**	Counties vary from one year to another
Year, State, School District	250,231	3.2 **MB**	School District vary from one year to another
Year, State/County, Geo Indicator, Age Category, Race Category, Gender Category, Income Category	1,753,926	82 **MB**	Certain category values vary across years

Table 1. Unique combinations of variables sets

Processing such amounts of unique combinations at run time resulted in unsatisfactory user experience! Promoting the entire application artifacts (code, data, images, files, etc.) between environments (Dev, Test, Prod) are impacted by the footprint of these lookup tables.
I had to find an alternative approach to the traditional OLAP approach in order to reduce the size of the combinations without affecting the integrity of the data and the relationship amongst the values of the variables.

SOLUTION

This is where the power of the SAS language came to the rescue, and provided me with straightforward processing techniques allowed me to achieve my goals.

1. SOLVING WEB APPLICATION DATA SIZE

Having

- SAS supports long character strings (32,767 chars)
- SAS provides first. & last. processing
- All the variables I had to deal with have relatively short code values
- The ability to develop custom reusable data manipulation macros using the SAS macro programming language

This allowed me to transpose and collapse particular variable values into a single space delimited string, and find unique combinations based on the newly created string value.

The following screen shots illustrate the data transformation.

VIEWTABLE: Work_r.Saipesd

	State Fips Code	District ID	Estimate Year	_TYPE_	_FREQ_
1	01	0100001	1995	7	1
2	01	0100001	1997	7	1
3	01	0100001	1999	7	1
4	01	0100001	2000	7	1
5	01	0100001	2001	7	1
6	01	0100001	2002	7	1
7	01	0100001	2003	7	1
8	01	0100001	2004	7	1
9	01	0100001	2005	7	1
10	01	0100001	2006	7	1
11	01	0100001	2007	7	1
12	01	0100001	2008	7	1
13	01	0100001	2009	7	1
14	01	0100001	2010	7	1
15	01	0100003	1995	7	1
16	01	0100003	1997	7	1
17	01	0100003	1999	7	1
18	01	0100003	2000	7	1
19	01	0100003	2001	7	1
20	01	0100003	2002	7	1

Display 1. Standard PROC SUMMARY output data set with 250,231 observations

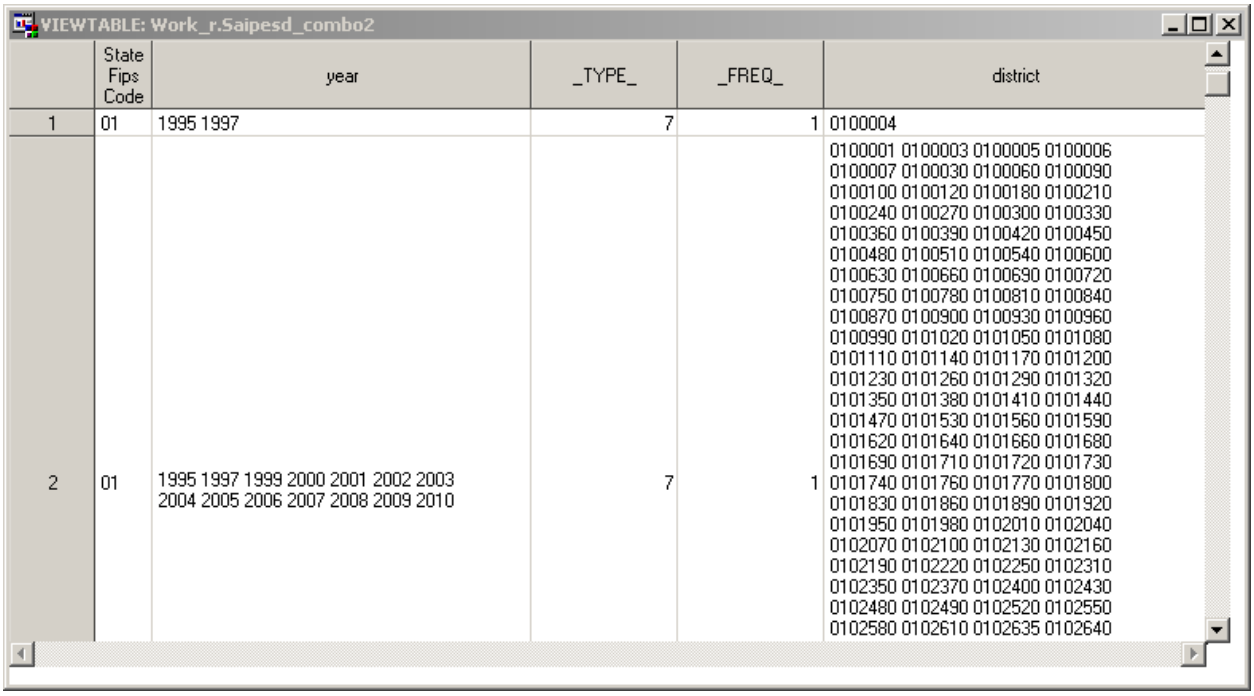
By transposing the Year values while maintaining the State and School District values, I got the following output data set

VIEWTABLE: Work_r.Saipesd_combo

	State Fips Code	District ID	_TYPE_	_FREQ_	year
1	01	0100001	7	1	1995 1997 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010
2	01	0100003	7	1	1995 1997 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010
3	01	0100004	7	1	1995 1997
4	01	0100005	7	1	1995 1997 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010
5	01	0100006	7	1	1995 1997 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010
6	01	0100007	7	1	1995 1997 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010
7	01	0100008	7	1	1997 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010
8	01	0100011	7	1	2001 2002 2003 2004 2005 2006 2007 2008 2009 2010
9	01	0100012	7	1	2004 2005 2006 2007 2008 2009 2010
10	01	0100013	7	1	2005 2006 2007 2008 2009 2010
11	01	0100030	7	1	1995 1997 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010
12	01	0100060	7	1	1995 1997 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010
13	01	0100090	7	1	1995 1997 1999 2000 2001 2002 2003

Display 2. Reducing 250,231 observations down to 14,837 observations.

Taking this one-step further, by transposing School District values while maintaining the State and Year gave me this output data set



VIEWTABLE: Work_r.Saipesd_combo2

	State Fips Code	year	_TYPE_	_FREQ_	district
1	01	1995 1997	7	1	0100004
2	01	1995 1997 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010	7	1	0100001 0100003 0100005 0100006 0100007 0100030 0100060 0100090 0100100 0100120 0100180 0100210 0100240 0100270 0100300 0100330 0100360 0100390 0100420 0100450 0100480 0100510 0100540 0100600 0100630 0100660 0100690 0100720 0100750 0100780 0100810 0100840 0100870 0100900 0100930 0100960 0100990 0101020 0101050 0101080 0101110 0101140 0101170 0101200 0101230 0101260 0101290 0101320 0101350 0101380 0101410 0101440 0101470 0101530 0101560 0101590 0101620 0101640 0101660 0101680 0101690 0101710 0101720 0101730 0101740 0101760 0101770 0101800 0101830 0101860 0101890 0101920 0101950 0101980 0102010 0102040 0102070 0102100 0102130 0102160 0102190 0102220 0102250 0102310 0102350 0102370 0102400 0102430 0102480 0102490 0102520 0102550 0102580 0102610 0102635 0102640

Display 3. Reducing 14,837 observations down to 364 observations only. Bingo!

Applying the same techniques against the other combination tables yielded the following results

Variables Set	# Unique Combinations	Disk Usage	# Unique Combinations after Transposing Values	Disk Usage after Transposing Values	Reductions %
Year, State/County	70,239	640 **KB**	**13**	**136 KB**	**99.98%, 78.75%**
Year, State, School District	250,231	3.2 **MB**	**364**	**264 KB**	**99.85%, 91.94%**
Year, State/County, Geo Indicator, Age Category, Race Category, Gender Category, Income Category	1,753,926	82 **MB**	**58**	**328 KB**	**100%, 99.61%**

Table 2. Transposed unique combinations of variables sets

With combinations in such low numbers, we were able to maintain optimum application initialization and run time processing.

CONCLUSION

Working with large data sets often requires adoption of alternative techniques beyond compression and other standard functionalities provided by SAS.

I would strongly encourage you all to think outside of the box and find ways to innovate. After all, developing custom solutions can sometime be frustrating and demanding, but when they work, they can be very rewarding.

REFERENCES

ACKNOWLEDGMENTS

CONTACT INFORMATION

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

Name: Ahmed Al-Attar
Organization: AnA Data Warehousing Consulting, LLC
City, State: McLean, VA
Work Phone: 703-477-7972
Email: ahmed.al-attar@anadwc.com
Web: http://www.anadwc.com

APPENDIX

```
/* ---------------------------------------------------------------------- */
/* Macro program to process combination data set and */
/* generate custom transposed output version.              */
/* ---------------------------------------------------------------------- */
%MACRO shrinkCombos(
          p_inDsName=              /* Input Source Data Source */
        , p_outDsName=             /* Output Data Set */
        , p_byClause=              /* Classification Variables Set */
        , p_classVar=              /* Class Variable to Transpose */
        , p_prntClassVar=          /* Preceding Class Variable */
        , p_aggrVarLen=10          /* Variable length of the new transposed values */
        );
        /* Sort the Data based on specified class clause */
        PROC SORT DATA=&p_inDsName;
                BY        &p_byClause;
        RUN;

        /* Process the data and generate the transposed value */
        DATA &p_outDsName(DROP=&p_classVar COMPRESS=YES
                                        RENAME=(aggr=&p_classVar));
                SET &p_inDsName;
                BY &p_byClause;

                LENGTH aggr $&p_aggrVarLen ;
                RETAIN aggr ;

                IF (FIRST.&p_prntClassVar) THEN
                        aggr='';

                aggr = catx(' ',aggr,&p_classVar);

                IF (LAST.&p_prntClassVar) THEN
                        OUTPUT;
        RUN;
%MEND shrinkCombos;

/* ---------------------------------------------------------------------- */
/* Macro program to convert comma separated values */
/* into multiplication of INDEXW function calls.           */
/* ---------------------------------------------------------------------- */
%MACRO multipleConditions(p_values=, p_varName=);
        %local l_condition l_valCount;
        %let l_valCount = %eval(%sysfunc(countc(%superq(p_values),%str(,)))+1);
        %let l_condition = (INDEXW(&p_varName,
%str(%')%sysfunc(tranwrd(%superq(p_values),%str(,),%str(%'%)*INDEXW%(&p_varName, %')))%str(%'))) GE
&l_valCount ;
        /* %put &=l_condition; */
        %unquote(&l_condition)
%MEND multipleConditions;
```

```
/* ------------------------------------------------------------------ */
/* Macro program to parse out space separated values */
/* into a unique list using an intermediate Hash Object */
/* ------------------------------------------------------------------ */
%MACRO getUniqueList(p_inDsName=, p_varName=, p_outDsName=);
	DATA _NULL_;
	 SET &p_inDsName(KEEP=&p_varName RENAME=(&p_varName=&p_varName._orig)) end=last;

	 /* Declare a Hash Object to hold unique values */
	 if (_n_=1) then
	 do;
		declare hash ho();
		rc=ho.definekey("&p_varName");
		rc=ho.defineDone();
	 end;

	 i=1;

	 /* Parse out the space separated values */
	 DO UNTIL(SCAN(&p_varName._orig,i,' ') EQ '');
		&p_varName = SCAN(&p_varName._orig,i,' ');

		/* If value does not exist, add it */
		if (ho.find()NE 0) then
			ho.add();

		i+1;
	 END;

	 /* Output the final unique list */
	 if (last) then
		ho.output(dataset:"&p_outDsName");
	RUN;
%MEND getUniqueList;

/****************/
/* Usage Examples */
/****************/
PROC SUMMARY DATA=saipe.saipeschldstrct(KEEP=year state district) NWAY;
	CLASS	state district year;
	OUTPUT OUT=sgf.saipeschldstrct_summary(Drop=_:);
RUN;

* Apply required manipulations to the list of unique combinations;
%shrinkCombos(p_inDsName=sgf.saipeschldstrct_summary, p_outDsName=work.summary2
, p_byClause=%str(state district year), p_classVar=year, p_prntClassVar=district, p_aggrVarLen=300);

%shrinkCombos(p_inDsName=work.summary2, p_outDsName=sgf.saipeschldstrct_combolkup
, p_byClause=%str(state year district), p_classVar=district, p_prntClassVar=year, p_aggrVarLen=15000);
```

```
/* ========--------- With Filtering - Selective values  ---------======== */
DATA want;
        SET sgf.saipeschldstrct_combolkup;
        WHERE %multipleConditions(p_values=%str(2011,2013), p_varName=year)
        AND state IN ('02','04');
RUN;
%getUniqueList(p_inDsName=want, p_varName=district, p_outDsName=work.test);
```