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.

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:

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

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.
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.

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.
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:

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

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

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.
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;
%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(N=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);
DATA want;
    SET sgf.saipechldistrict_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);