SAS® GLOBAL FORUM 2020 #SASGF

Paper 4978-2020

Optimal Use of Extended Data Types, Memory Mapping in SAS® Viya® 3.3

Saurabh Tripathi, Pranay Barua, Ayush Tiwari, Core Compete

ABSTRACT

The major challenges for business users these days, are optimally ingesting data in terms of size, time and performance. This paper focuses on optimal use of SAS® Cloud Analytic Services (CAS), and how SAS® Viya® users will be benefitted using a set of rules to achieve the goals in the most efficient manner. We developed a cognitive approach of data ingestion in SAS® Viya®, which can handle the data bloating and load time that in effect improves the performance of SAS® Visual Analytics on SAS® Viya®. Have you ever encountered problems such as data bloating while loading to CAS or experienced that the loading time was more than what was expected or encountered a problem while accessing reports in SAS® Viya® Visual Analytics? To overcome these problems, we have leveraged capabilities of SAS® Viya® like powerful new table indexing, block mapping, memory mapping, extended data types, native data formats capabilities that have the potential to significantly improve performance for data handling and analytics actions submitted to CAS. This paper argues for performance benefits obtained using mentioned approaches example data get compressed from 170 GB to 60 GB, CPU and elapsed time reductions reported by measuring report performances, the CPU times could be up to three orders of magnitude greater than without mapping, which in effect reduces the operational cost.

The following examples are included:

- Optimal use of extended data types
- Optimal use of hdat conversion
- Memory -mapping and block mapping
- Indexing of appropriate variables while loading data into CAS
- Indexing in SAS® Viya® v/s SAS 9.4
- Handling special character’s in the data
- Performance comparisons

Keywords: Memory mapping, block mapping, indexing, SAS Viya performance optimization

INTRODUCTION

SAS® Cloud Analytic Services (CAS) provides high performance with its multi process power introduced in SAS Viya architecture.

The following major topics are discussed:

- Extended data types and its optimal use to utilize the memory efficiently.
- Hdat file format to achieve memory and block mapping.
- SAS® Cloud Analytic Services (CAS) table indexing to improve report performance.
- Performance comparisons

EXTENDED DATA TYPES

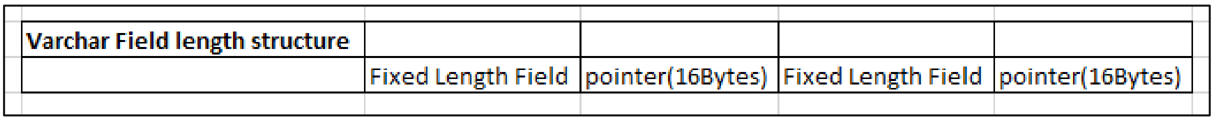
Default encoding in SAS® Cloud Analytic Services (CAS) is UTF-8. UTF-8 is a variable width encoding system. When source data is not encoded as UTF-8, the data may require more space in CAS as compared to source system. In order to overcome the expansion of data size, CAS offers the NCHARMULTIPLIER parameters to define how much to **expand character fields** to avoid truncation.

NCHARMULTIPLIER=n   *where* 0 < *n* ≤ 4

Character field in a SAS® Cloud Analytic Services (CAS) varies and it could be 4 times as big as the length in the source data. Example, a 20 GB table having character variables may become 80 GB after loading it to SAS® Cloud Analytic Services (CAS).

SAS® Cloud Analytic Services (CAS) comes with multiple data types unlike BASE SAS where we have only character or numeric as data type. Varchar not only help to save storage it also improves the performance by reducing the size of record being processed.

Apart from extended data types, SAS® Cloud Analytic Services (CAS) also supports User defined formats (UDFs) , it servs the purpose of display formatting on raw feeds. This also plays a role in reducing the size of records and as a result reducing the data size. Extended data types and User defined formats (UDFs) lead to performance gains.



Varchar Field length structure				
	Fixed Length Field	pointer(16Bytes)	Fixed Length Field	pointer(16Bytes)

Figure 1: Varchar Variable Description

Varchar is a wise choice for character columns in order to reduce data size, but it should be a careful decision otherwise we will end up in increasing the data size. It's highly recommended that VARCHAR should be used on characters with more lengths associated with it and could vary in length, e.g. description. If VARCHAR is used wisely will result in saving lot of space. We need to analyze the length and accordingly should choose which should be taken as VARCHAR. It assigns the length in the following way. Hence it is recommended to use mix of CHAR-VARCHAR.

The best way is to have Char-Varchar definition using the below conditions:

- If the length of column exceeds 30, we should go for Varchar data type. The reason being the length of VARCHAR variables is determined based on the number of characters the string contains. The length of CHAR variables is determined based on the number of bytes the characters in the string requires
- If the length of column is smaller better go with Char datatype. The reason being for smaller length we will pay addition penalty of 16 BYTES for Varchar pointer
- If you directly import a file, it will have all character variables defined as Varchar

Constraints

Char-varchar can only be applied with import files or data prepared in SAS® Cloud Analytic Services (CAS).

The VARCHAR data type is not supported by the SAS V9 engine. Therefore, you must use a CAS engine libref on the output table when creating a VARCHAR.

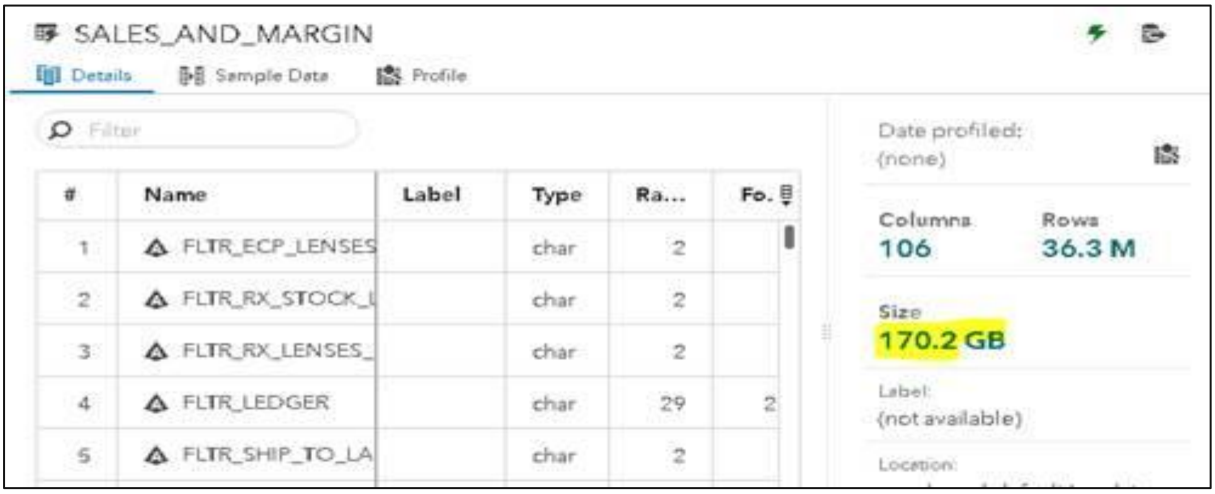
Importing data with Char-Varchar definition

```
options cashost="server name" casport=XXXX;

cas;

cas mySession sessopts=(caslib=libname timeout=1800 locale="en_US");


libname tmp 'Path';

libname tmp cas;
```

```
proc casutil;
load casdata="XXX.csv"  casout="XXXX"                          /* 2 */
        importoptions=(filetype="csv" getnames="true"  varChars="FALSE"
vars=(
   (name="Column1="CHAR", length=10  ),
(name="Column2" type="VARCHAR"  ),
(name="Column3" type="CHAR" length=10 )
) ) promote                         /* 5 */
       label="Fact table for User-to-Item Analysis"
;
quit;
```

Impact of Char-Varchar conversion

A. With Char definition



SALES_AND_MARGIN

#	Name	Label	Type	Ra...	Fo.
1	FLTR_ECP_LENSES		char	2	
2	FLTR_RX_STOCK_L		char	2	
3	FLTR_RX_LENSES_		char	2	
4	FLTR_LEDGER		char	29	2
5	FLTR_SHIP_TO_LA		char	2	

Date profiled:
(none)

Columns 106
Rows 36.3 M

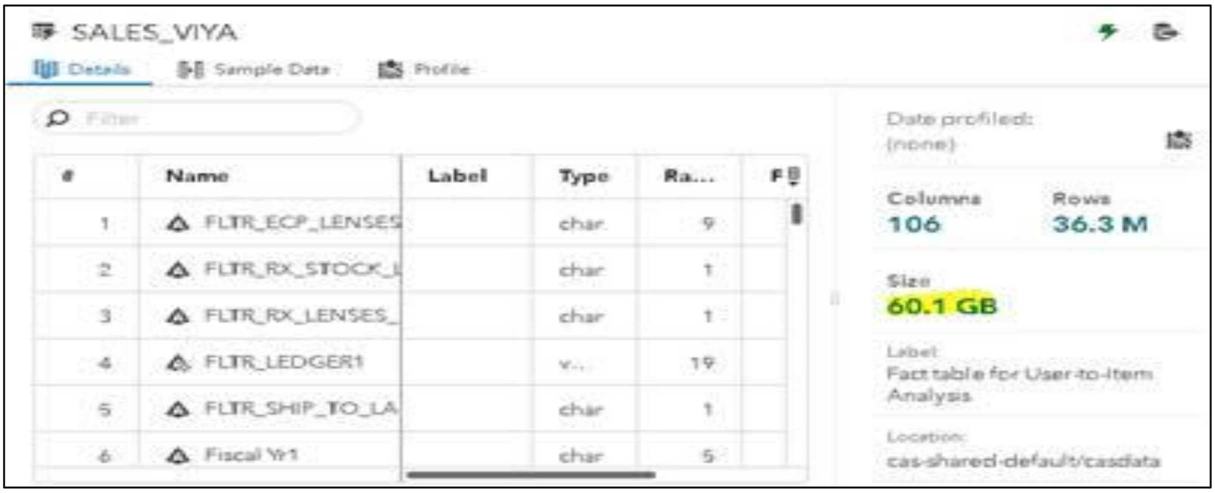
Size
170.2 GB

Label:
(not available)

Location:

Figure 2: With Char Definition

B. With Char-Varchar definition



SALES_VIYA

#	Name	Label	Type	Ra...	F
1	FLTR_ECP_LENSES		char	9	
2	FLTR_RX_STOCK_L		char	1	
3	FLTR_RX_LENSES_		char	1	
4	FLTR_LEDGER1		v...	19	
5	FLTR_SHIP_TO_LA		char	1	
6	Fiscal Yr1		char	5	

Date profiled:
(none)

Columns 106
Rows 36.3 M

Size
60.1 GB

Label:
Fact table for User-to-Item Analysis

Location:
cas-shared-default/casdata

Figure 3: With Char- Varchar Definition

BLOCK AND MEMORY MAPPING

SAS® Cloud Analytic Services (CAS) table consists of physical data segments called Large Blocks. These physical data segments are further divided into smaller blocks. Each Small Block holds a contiguous which holds values of columns defined in the table. The varying-length data values for any VARCHAR or VARBINARY column types occupy a separate area following the rows. When varying-length column types are defined in the table, each row contains corresponding references to its values in the varying-length data area.

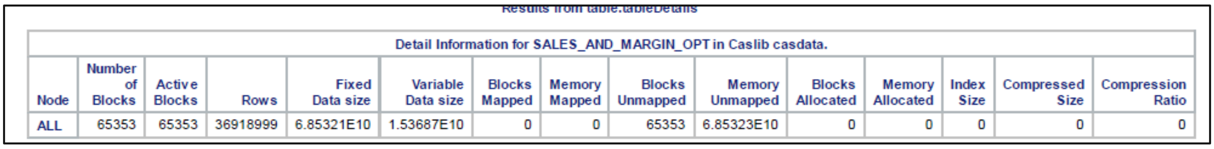
CAS can run in Symmetric Multiprocessing SMP or Massively Parallel Processing MMP. In either process CAS processes distributed data on multiple threads.

For SMP having single machine or for distributed server file source type doesn't use *CAS_DISK_CACHE*. In such cases memory mapping is already in place and hence keeping a check on memory utilization.

Number of backup copies will depend on time period for which table is supposed to be used. If a large output table is used for short period, in such cases programmers can set copies option to zero.
But we need to be very careful while setting this option. If redundant blocks are not available, there are high chances of no-fault tolerance.

sashdat is native data file type for CAS. Hence while loading a dataset it recommends converting this to sashdat it may not have impact on data size, but it will save loading time. Conversion script is given below:

```
proc casutil;
   load casdata="dataset.sashdat"
        importoptions=(filetype="hdat" );
run;
```

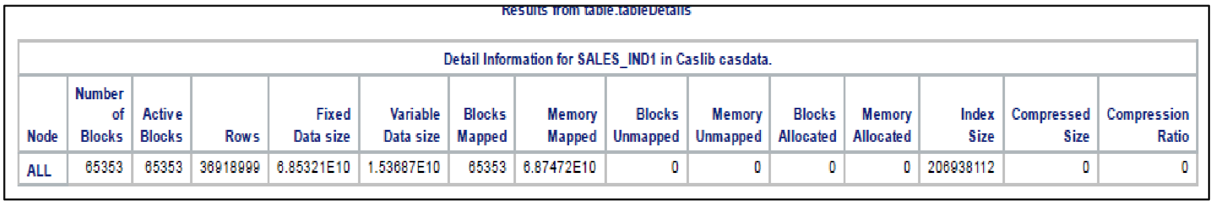


Results from table.tableDetails

Detail Information for SALES_AND_MARGIN_OPT in Caslib casdata.

Node	Number of Blocks	Active Blocks	Rows	Fixed Data size	Variable Data size	Blocks Mapped	Memory Mapped	Blocks Unmapped	Memory Unmapped	Blocks Allocated	Memory Allocated	Index Size	Compressed Size	Compression Ratio
ALL	65353	65353	36918999	6.85321E10	1.53687E10	0	0	65353	6.85323E10	0	0	0	0	0

Figure 4: Example dataset without memory and block mapping



Results from table.tableDetails

Detail Information for SALES_IND1 in Caslib casdata.

Node	Number of Blocks	Active Blocks	Rows	Fixed Data size	Variable Data size	Blocks Mapped	Memory Mapped	Blocks Unmapped	Memory Unmapped	Blocks Allocated	Memory Allocated	Index Size	Compressed Size	Compression Ratio
ALL	65353	65353	36918999	6.85321E10	1.53687E10	65353	6.87472E10	0	0	0	0	206938112	0	0

Figure 5: Example dataset with memory and block mapping

INDEXING

An index on a CAS table can be created on multiple columns if required, however it's important to choose a column on which we should create an index. In latest releases its working on extended data types as well.

Now if we go into depth to understand how indexing improves performance it works as the indexed column will have a range of values from MIN to MAX defined for a large block. Once the user defines a subset over the data it will first look for the range in which the desired value falls and it will pick large block ignoring the rest of the large blocks. We save lot of time by following the approach of discarding the options to search for the smaller blocks among number of large blocks. This results in decreasing the CPU and real time for subsetting the data.

Unlike Base SAS where for each .sas7bdat dataset if an index is created then a separate index file used to get stored as .sas7bndx, in CAS if an index is created on columns it will not be written to a separate file instead these are already part of large blocks and written to underlying storage to use the space efficiently.

Below is the illustrated example of how indexing in CAS improves the efficiency of sub setting. Let's say we have a column *company_id*. In the case of no indexing for anywhere condition on *company_id* entire table needs table needs to be scanned as compared to when an index is created on column *company_id* it will split it into number of large blocks and smaller blocks within large blocks. When an indexing is available, for any sub setting it will directly go to the bigger block matching its filter condition and same applies for smaller blocks as well.

Without Indexing on company_id:

No indexing
1……n+500

With Indexing on company_id:

Large Block1…500
Small Block 1…100
Small Block 101…200
Small Block 201…300
Small Block 301…400
Small Block 401…500

Large Block 501…1000
Small Block 501…600
Small Block 601…700
Small Block 701…800
Small Block 801…900

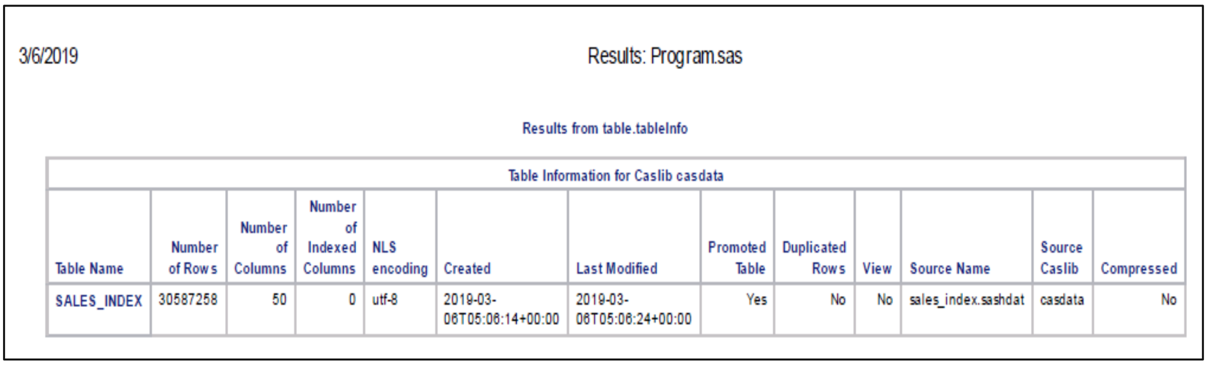
Small Block 901…1000

Large Block 1001…1500
Small Block 1001…1100
Small Block 1101…1200
Small Block 1201…1300
Small Block 1301…1400
Small Block 1401…1500

Large Block n-500 n
Small Block n…n+100
Small Block n+101…n+200
Small Block n+201…n+300
Small Block n+301…n+400
Small Block n+401…n+500

A. Extracting table information

```
proc cas;

table.tableInfo / name='SALES_INDEX';

run;
```



3/6/2019

Results: Program.sas

Results from table.tableInfo

Table Information for Caslib casdata

Table Name	Number of Rows	Number of Columns	Number of Indexed Columns	NLS encoding	Created	Last Modified	Promoted Table	Duplicated Rows	View	Source Name	Source Caslib	Compressed
SALES_INDEX	30587258	50	0	utf-8	2019-03-06T05:06:14+00:00	2019-03-06T05:06:24+00:00	Yes	No	No	sales_index.sashdat	casdata	No

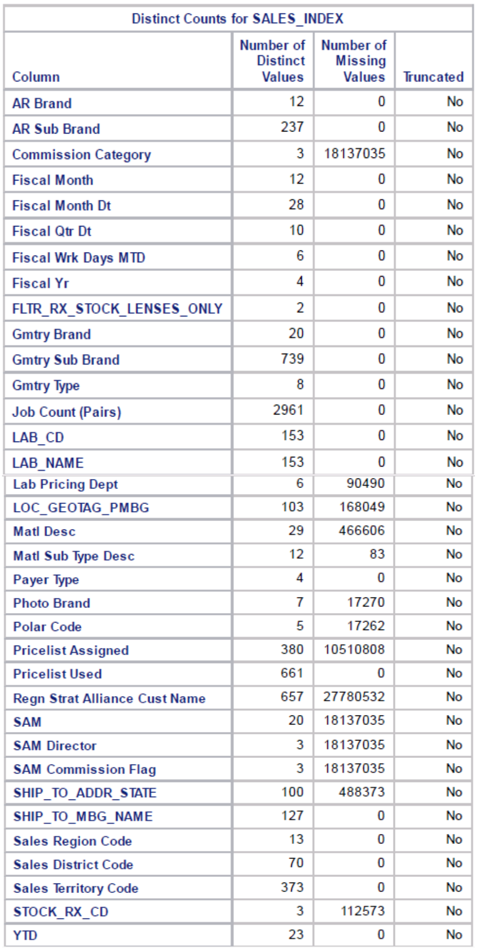
Figure 6: Output table information

B. Column information

```
proc cas;

simple.distinct / table={name="SALES_INDEX"};run;
```

C. Output: Distinct Counts

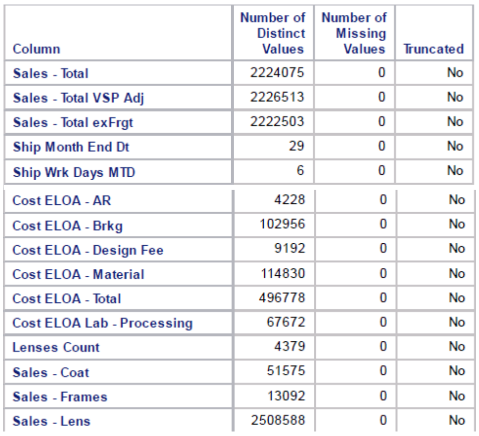
Extract different dimensions related to a table e.g. distinct value count, missing value count etc.



Distinct Counts for SALES_INDEX

Column	Number of Distinct Values	Number of Missing Values	Truncated
AR Brand	12	0	No
AR Sub Brand	237	0	No
Commission Category	3	18137035	No
Fiscal Month	12	0	No
Fiscal Month Dt	28	0	No
Fiscal Qtr Dt	10	0	No
Fiscal Wrk Days MTD	6	0	No
Fiscal Yr	4	0	No
FLTR_RX_STOCK_LENSES_ONLY	2	0	No
Gmtry Brand	20	0	No
Gmtry Sub Brand	739	0	No
Gmtry Type	8	0	No
Job Count (Pairs)	2961	0	No
LAB_CD	153	0	No
LAB_NAME	153	0	No
Lab Pricing Dept	6	90490	No
LOC_GEOTAG_PMBG	103	168049	No
Matl Desc	29	466606	No
Matl Sub Type Desc	12	83	No
Payer Type	4	0	No
Photo Brand	7	17270	No
Polar Code	5	17262	No
Pricelist Assigned	380	10510808	No
Pricelist Used	661	0	No
Regn Strat Alliance Cust Name	657	27780532	No
SAM	20	18137035	No
SAM Director	3	18137035	No
SAM Commission Flag	3	18137035	No
SHIP_TO_ADDR_STATE	100	488373	No
SHIP_TO_MBG_NAME	127	0	No
Sales Region Code	13	0	No
Sales District Code	70	0	No
Sales Territory Code	373	0	No
STOCK_RX_CD	3	112573	No
YTD	23	0	No

Figure 7.1: SAS Viya Output



Column	Number of Distinct Values	Number of Missing Values	Truncated
Sales - Total	2224075	0	No
Sales - Total VSP Adj	2226513	0	No
Sales - Total exFrgt	2222503	0	No
Ship Month End Dt	29	0	No
Ship Wrk Days MTD	6	0	No
Cost ELOA - AR	4228	0	No
Cost ELOA - Brkg	102956	0	No
Cost ELOA - Design Fee	9192	0	No
Cost ELOA - Material	114830	0	No
Cost ELOA - Total	496778	0	No
Cost ELOA Lab - Processing	67672	0	No
Lenses Count	4379	0	No
Sales - Coat	51575	0	No
Sales - Frames	13092	0	No
Sales - Lens	2508588	0	No

Figure 7.2: SAS Viya Output

D. Output: Indexing on *LAB_NAME*

```
Proc cas;

table.index/casout={indexVars={"LAB_NAME"},name="SALES_INDEXED”}
table={name="SALES_INDEX"};

run;
```

E. Performance Test

- Without Index:

```
  data test1;

  set casdata.sales_index;

where 'Regn Strat Alliance Cust Name'n="Norris & Kelly, PA dba Family Eyecare Center";

  run;
```

Note: The data set WORK.TEST1 has 7184 observations and 50 variables.

Note: DATA statement used (Total process time):

```
real time        5.93 seconds

cpu time         0.83 seconds
```

- With Index:

```
data test2;
set casdata.IND2;
where 'Regn Strat Alliance Cust Name'n="Norris & Kelly, PA dba Family Eyecare Center";
 run;
```

Note: The data set WORK.TEST2 has 7184 observations and 50 variables.

Note: DATA statement used (Total process time):

```
real time      0.64 seconds
cpu time       0.54 seconds
```

F. Multi-level performance testing

1. 'Gmtry Sub Brand'n

Gmtry Sub Brand	Without Index	With Index
Real Time	01:40.0	56.37 seconds
CPU Time	21.54 seconds	19.97 seconds

2. "*LAB_NAME*" "Regn Strat Alliance Cust Name" "Gmtry Sub Brand" "Pricelist Used

"LAB_NAME" "Regn Strat Alliance Cust Name" "Gmtry Sub Brand" "Pricelist Used	Without Index	With Index	Filter
Real Time	5.93 seconds	0.64 seconds	'Regn Strat Alliance Cust Name'n="Norris & Kelly, PA dba Family Eyecare Center" and 'Gmtry Sub Brand'n in("BI/TRIFOCAL","CONTACTS"
CPU Time	0.83 seconds	0.54 seconds	

CONCLUSIONS

This paper concludes the advantages of CAS table indexing, extended data types, memory and block mapping supported in SAS Viya 3.3. The major objectives to document it are as follows:

- Wise choice of extended datatypes to reduce data ballooning or data truncation while loading it to CAS.
- Using the concept of memory and block mapping by having the data in the native sashdat format.
- Using the indexing to improve subsetting of data and report performance.
- Measurable performance gains by reducing the time required to query or subset the data with the help of indexes.

REFERENCES

https://documentation.sas.com/?docsetId=casfun&docsetTarget=n16qbskv0hwfq1n1lnrqq605p6sv.htm&docsetVersion=3.5&locale=en

CONTACT INFORMATION

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

Saurabh, Core Compete
E-mail: saurabh.tripathi@corecompete.com
www.corecompete.com