

Dictionary.columns is your friend while appending or moving data

Kiran Venna, Dataspace Inc.

ABSTRACT

Dictionary.columns is a dictionary table, which gives metadata information of columns. Dictionary.columns is very helpful in understanding various data related issues. Data integrity issues, while appending data from external files to SAS® Datasets can be understood with help of Dictionary.columns. Appropriate appending of external files to SAS Datasets can be done with help of SAS Macros. Data movement from Hive® to Teradata® by using SAS can cause an issue when variable name length is greater than 32. Longer variable name issues can be evaluated with the aid of Dictionary.columns. Proc FEDSQL can solve variable names length issue. While inserting data from SAS to RDBMS, difference in number of columns can cause an issue and this issue can be analyzed and fixed with help of dictionary.columns.

INTRODUCTION

SAS maintains system tables to store metadata information about all the objects related to SAS session. These tables are read only tables and belong to libref dictionary and provide metadata information to the user. Libref dictionary has total of 32 tables, which give information about catalogs, options, formats, tables, columns, macros, members and others. The query below gives information about all tables with their description available in libref dictionary.

```
proc sql;
  select distinct memname,
                 memlabel
  from          dictionary.dictionaries;
quit;
```

Partial output of above query is shown below

Member Name	Data Set Label
CATALOGS	Catalogs and catalog-specific information
CHECK_CONSTRAINTS	Check constraints
COLUMNS	Columns from every table
CONSTRAINT_COLUMN_USAGE	Constraint column usage
CONSTRAINT_TABLE_USAGE	Constraint table usage
DATAITEMS	Information Map Data Items

In this paper main focus is to discuss on dictionary.columns table. Dictionary.columns provides lot of metadata information about columns, like data type of column, position of the column along with other information like format and label of columns. The below query, will help us to know what kind of information is available in dictionary.columns.

```
proc sql;
  describe table dictionary.columns;
quit;
```

Partial result of the above query is shown below.

```
Create table DICTIONARY.COLUMNS
(
  libname char(8) label='Library Name',
  memname char(32) label='Member Name',
  memtype char(8) label='Member Type',
  name char(32) label='Column Name',
  type char(4) label='Column Type',
  length num label='Column Length',
  npos num label='Column Position',
  varnum num label='Column Number in Table',
  label char(256) label='Column Label',
  format char(49) label='Column Format');
```

Day-to-day life of SAS Programmer usually involves moving or appending the data from external files or from DBMS into SAS and vice versa. Data movement from/into SAS from external files or other DBMS can pose certain issues. These issues can be better understood with help of dictionary.columns and is topic of this paper.

Data Movement and Append issues

Appending data or moving data with column metadata mismatches between a source file/table and target table can result in a SAS job failure or a data integrity issue. When tables are really wide with lot of columns, it is very difficult to find which or how many columns are having metadata mismatches. This column mismatches can be due to difference in column type, column format, column position or others. Mismatched columns between source and target tables can be easily identified by dictionary.columns. Moving data between 2 different DBMS via SAS/Access can pose unique problems due to maximum column name length restriction of 32 in SAS as opposed to longer column name length allowed in other DBMS. Column name length issues can be understood with help of dictionary.columns. In this paper, I will discuss three different issues related to data moving or appending data by SAS and how this issues are identified with help of dictionary.columns. Solutions for these 3 issues by using SAS resources will also be discussed.

Topics covered in this paper are

1. Appending of wrong data due to column order issue with external file.
2. Issue of longer column name length, while moving data from Hive to Teradata with help of SAS.
3. Difference in number of columns issue, while appending data from SAS to Teradata.

1. Appending of wrong data due to column order issue with external file

Here's a case study of wrong data appending to the production data due to column order issue with external file.

A general practice in this below sample production Job is to read daily external files by using infile statement followed by an input statement. This gives tremendous control for defining proper length, formats and other properties of the column. Even though this approach has advantage but can cause issues of wrong data getting appended, when the order of variables is changed in external file. For a standard file reading generally below code is used and this code reads the data exactly as intended.

```
data WORK.PERSON;  
  INFILE "&filenm." firstobs=2 LRECL=10240 missover dsd;  
  INPUT fname $ lname $ state $ imp ;  
run ;
```

Output of data, when everything is read properly is shown below.

fname	lname	state	imp
James	Smith	KY	200
Maria	Rodriguez	NY	300

But when the data is in organized in different fashion in external file, which can happen rarely in production environment can lead to data integrity issue.

For example, when data is in external file is organized as shown below.

fname	imp	lname	state
Thomas	100	Miller	NY

After reading above data with input statement data appears as shown below.

fname	lname	state	imp
Thomas	100	Mi	.

In production after external file is read, data is appended to the permanent dataset by using below code.

```
proc append base =PROD.PERSON
  data =PERSON1;
run;
```

This will lead to incorrect data getting appended and an example of the same is shown below.

fname	lname	state	imp
James	Smith	KY	200
Maria	Rodriguez	NY	300
Thomas	100	Mi	.

Understanding of incorrect data append from external files by dictionary.columns and fixing it

Main idea here is to compare column order of production table (or reference table) with column order of new external file. Broadly this can be done in three steps.

First step is to create a macro variable containing column names from production table (or reference table) as shown below.

```
proc sql noprint;
select name into :standard separated by ','
from dictionary.columns
where upper(libname) = 'PROD'
and upper(memname) = 'PERSON';
quit;
```

Second step is to read column names from external files, which is done by creating a work table with zero record with column names. Once work table is created, column names are placed in new macrovariable. Zero record work table can be created from external file is by using options obs =1 and then by using proc import with getnames =yes option as shown below.

```
options obs=1;
PROC IMPORT OUT= WORK.PERSON1 DATAFILE= "/somelocation/person1.csv"
DBMS=CSV REPLACE;
GETNAMES=YES;
DATAROW=2;
RUN;
```

Once dataset is created from external file with zero records, macro variable containing column names are created using below query.

```
proc sql noprint;
select name into :test separated by ','
from dictionary.columns
where upper(libname) = 'WORK'
and upper(memname) = 'PERSON1';
quit;
```

Third step is to compare column name order of reference/production table with newly created zero record dataset. Appending to production table is done conditionally only when column orders match, else when column order is not matched should be notified to required personnel. The below macro demonstrates this point.

```
%macro abc;
  Proc options obs =max;
  %if "&standard" = "&test "
  %then %do;
    DATA WORK.PERSON;
      INFILE "&filenm." firstobs=2 LRECL=10240 missover dsd;
      INFORMAT fname $6. Iname : $9. state : $2. imp BEST32. ;
      INPUT fname $ Iname $ state $ imp ;
    RUN;
    Proc append;
      Base = PROD.PERSON
      Data = WORK.PERSON1;
    Quit;
  %end;
  %else %do;
    %put "Something is wrong please inform your manager";
  %mend abc;
```

2. Longer column length issue when moving data from Hive to Teradata by SAS

This scenario occurred, when an attempt was made to move data from Hive to Teradata. Below is an example of DATA step to move data from Hive to Teradata. Hivedb is libname for hive table and teradb is libname for Teradata table.

```
data teradb.shoppingmart;
  set hivedb.shoppingmart;
run;
```

When the above DATA step was executed it resulted in an error as shown below

```
ERROR: Variable shoppingmartactivitymodification has been defined as both character and
numeric
```

As a programmer, this error was initially difficult to understand and also quick search did not provide enough insights. After Analysis it was found that this was due to longer column names in Hive table.

Understanding of longer column name issue with dictionary.columns

Longer column name issue was caused due to maximum column name length allowed in hive (128), when compared to maximum column name length allowed in SAS (32). Whenever SAS is used, the column name from RDBMS or Hive which are longer than 32, are usually truncated to 32. In this case, there are two column names, which were greater than 32 in Hive. These columns were shoppingmartactivitymodificationdatetime (character variable) and shoppingmartactivitymodificationcustid (numeric variable). These two column names after truncation in SAS resulted in having same name and that is shoppingmartactivitymodification.

Usually column names of a Hive/RDBMS table are stored in both column **name** as well as in column **label** of dictionary.columns. In dictionary.columns, maximum allowed column **name** length is 32 and maximum allowed column **label** length is 256.

Hive column names in dictionary.columns column **name** will be truncated when column name length is greater than 32 but will never get truncated in column **label** of dictionary.columns as maximum column length for most of RDBMS/Hive is usually around 128.

Below query gives us information about columns, which were truncated but are not same.

```
proc sql;
  select  a.name, a.label
  from    dictionary.columns a
  inner join dictionary.columns b
  on      a.memname = 'SHOPPINGMART'
  and     a.libname = 'HIVE_DB'
  and     a.name = b.name
  and     a.label ne b.label;
quit;
```

Fixing longer column length issue with PROC FedSQL

PROC FedSQL can come to rescue, when dealing with longer column length issues. Advantage of PROC FedSQL is that when it connects to data source, it matches metadata of data source and target respectively and translates to source/target metadata without involving SAS/Access, so the truncation does not happen. PROC FedSQL code to move data from Hive to Teradata is shown below.

```
proc fedsq;
  create table teradb.SHOPPINGMART as
  select * from hive_db.SHOPPINGMART
quit;
```

3. Difference in number of columns while appending data from SAS to Teradata

Last case study in this paper is finding the difference in number of columns, while appending data from SAS to Teradata using insert statement in PROC SQL. When number of columns is not same for source

and target table during insert statement in PROC SQL, we get an error and results in failure of insert. If target table i.e. Teradata has more columns than SAS table it will result in error as shown below.

ERROR: Attempt to insert fewer columns than specified after the INSERT table name

Understanding the difference in numbers of columns issue with dictionary.columns

Insert statement in SQL are not dependent on column names, but are dependent on column positions of Source and target. Dictionary.columns has variable varnum, which indicates position of a variable in table. Comparison of varnum (column position) of source table with target table will gives us information of additional columns in target table.

For this particular case study, let us take into consideration, we are inserting data into Teradata table with extra columns and got the error as discussed earlier. Below code give us the information of all the additional columns that were not present in SAS table but were present in the Teradata table. In the below code, sasdb is libname for SAS table and teradb is libname for Teradata table.

```
proc sql;
  select * from dictionary.columns
  where memname = "EMP"
  and libname = 'TERADB'
  and varnum not in
      (select varnum from dictionary.columns
       where memname = "EMP"
       and upcase(libname) = 'SASDB');
quit;
```

Fixing difference in number of columns issue by using macrovariable

One approach to fix number of column issue is to create a macrovariable with name of columns whose positions between target and source match. This Macrovariable can be created from below code.

```
proc sql noprint;
  select name into :columns separated by ',' from dictionary.columns
  where memname = "EMP"
  and upcase(libname) = 'TERADB'
  and varnum in
      (select varnum from dictionary.columns
       where memname = "EMP"
       and libname = 'SASDB');
quit;
```

Once the macro variable is created as shown above. This macro variable can be used in the insert statement as shown in below code, which can facilitate insertion successfully.

```
proc sql;  
  insert into TERADB.EMP(&columns)  
  select * from SASDB.EMP;  
quit;
```

CONCLUSIONS

Dictionary.columns are very helpful in understanding columns of the tables available in any library. Dictionary.columns will give metadata information for columns of SAS tables along with other RDBMS tables. Comparing columns between various heterogeneous sources can be done with help of Dictionary.columns.

ACKNOWLEDGEMENTS

I would like to specially thank, Kirk Paul Lafler for encouraging me to write papers.

I would like to thank, Lakshmi Nadiya Chintalapudi, Charannag Devarapalli , Sarika Elisetti, Anvesh Reddy Perati, Srirama Reddy, Suryakiran Pothuraju for helping me with proof reading and giving their valuable suggestions.

REFERENCES

1. SAS/ACCESS® 9.4 for Relational Databases: Reference, Ninth Edition
<http://documentation.sas.com/api/docsets/acreldb/9.4/content/acreldb.pdf?locale=en#nameddest=titlepage>
2. Teradata Documentation
https://www.info.teradata.com/HTMLPubs/DB_TTU_15_00/index.html
3. SAS(R) 9.4 FedSQL Language Reference, Third Edition
<http://support.sas.com/documentation/cdl/en/fedsqlref/67364/PDF/default/fedsqlref.pdf>

CONTACT INFORMATION

Kiran Venna
kiranvenna@gmail.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.