Paper BB-04

QC Your SAS® and RDBMS Data Using Dictionary Tables

Harry Droogendyk, Stratia Consulting Inc., Lynden, ON

ABSTRACT

In the context of our daily occupations we are always examining data. Whether we're testing ETL processes that populate data marts, verifying data pulled for testing, or just becoming acquainted with unfamiliar data there are some rudimentary things we do typically do. Simple analysis of continuous variables such as min, max, mean etc... and frequency distributions of categorical variables are often used to provide quick insight.

This paper presents a macro that does the QC work for you, driving the process from dictionary tables, whether the data is from SAS® datasets or sourced from any of the DB systems accessible by SAS.

INTRODUCTION

What types of things do we typically do when we're seeking to understand either the shape or the quality of our data? Sometimes the interaction or relationship between variables is important but not always, or at least not initially. Often simple analysis will provide enough insight into the data to quickly identify quality issues or give us a good idea of the contents of the data set or table.

If simple analysis will suffice, we're left with a sleuthing exercise. Which of the character variables are really categorical? Are all the numeric variables continuous or are some of them discrete? Once those questions are answered, the appropriate SAS procedures can be run on the appropriate variables to create useful analysis.

Rather than having to go through the discovery exercise each time fresh data is encountered or new ETL results are to be verified, I developed a macro that will do the grunt work for me. The rest of the paper describes the methodology of the macro and how it makes the initial data QC exercise a very simple exercise. The macro is found in the appendix.

LAZY PROGRAMMERS USE DICTIONARY DATA

Lazy programmers are a good thing. Lazy programmers would rather put a little additional thought into the design of a solution and save themselves the subsequent maintenance that inflexible processes inevitably demand. Lazy programmers don't like hard-coding and would rather allow the data to drive the process. Doing so frees the lazy programmer from mundane tasks and allows time to be spent on much more useful tasks that add value and don't require mind-numbing maintenance.

DATA DRIVEN PROCESSES

Data driven processes use metadata, or the characteristics of the data itself, to determine the processing required. For instance, if the data contained only character columns or discrete numeric data, there would be no need to generate min, max, means etc… Metadata will provide the requisite information needed to create a data-driven QC process.

SURFACING METADATA

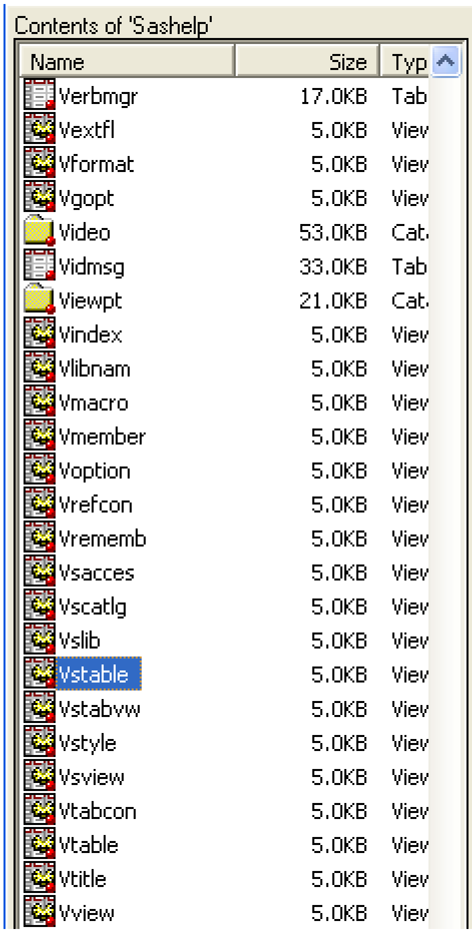
SAS has "dictionary" data that provide data *about the data*, or, metadata. The SAS dictionary data provides a great deal of insight into our SAS environment. For example, we can identify the libraries defined to the SAS session, data sets within those libraries and the columns within the data sets *and their characteristics*. Almost everything associated with a SAS session, whether it be batch or interactive, can be surfaced through the metadata. Additional dictionary information includes: assigned external files , macro variables, options, titles, formats etc…

The table and column metadata is very helpful for the QC exercise. Not only will the metadata provide the columns within the table, but also the types and lengths of each variable. Using metadata which is returned programmatically, the lazy programmer is able to build a process that utilizes the metadata to create a flexible process that will analyze the business data.

SAS metadata is available via three different avenues. PROC CONTENTS output can be directed to a SAS dataset and manipulated as required. Secondly, a special library named DICTIONARY is available for PROC SQL queries. DICTIONARY has a number of members, each relating to a different set of objects for which metadata is available,

eg. TABLES, COLUMNS, INDEXES etc… Thirdly, a series of SASHELP views are automatically available in any SAS session to both the data step and SQL procedure. SASHELP views have slightly different names than the DICTIONARY members, but they contain the same data. A sampling of SASHELP view names are VTABLE, VCOLUMN, VINDEX.

SASHELP Views



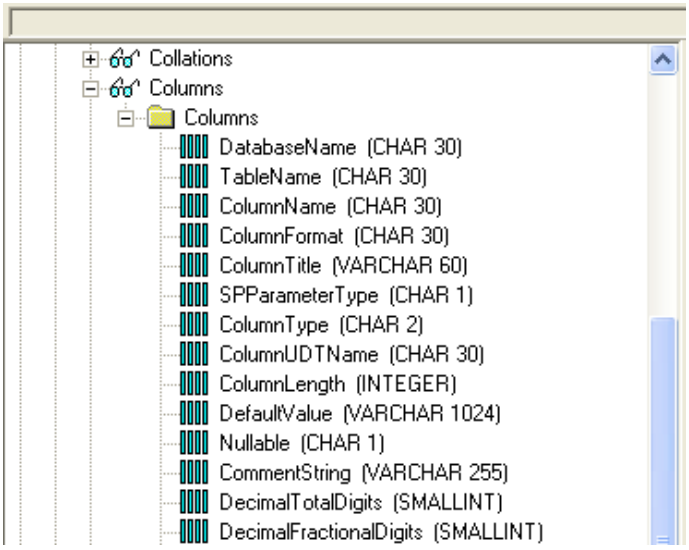
Querying **SAS** Dictionary Data:

```
proc sql;
 select libname, memname, name
   from sashelp.vcolumn
  where libname = 'WORK'
    and memname = 'CLASS'
  order by name
  ;
quit;
```

```
Library
Name      Member Name                       Column Name
ƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒƒ
WORK      CLASS                             Age
WORK      CLASS                             Height
WORK      CLASS                             Name
WORK      CLASS                             Sex
WORK      CLASS                             Weight
```

In the same way that SAS has dictionary data, relational database management systems (RDBMS) have similar facilities to provide metadata. Just as SAS has a set of tables or views to provide this data, each DB system also has a similar set of tables to surface the metadata. Unfortunately, each RDBMS uses different names for the dictionary table names and the column names within those dictionary tables.

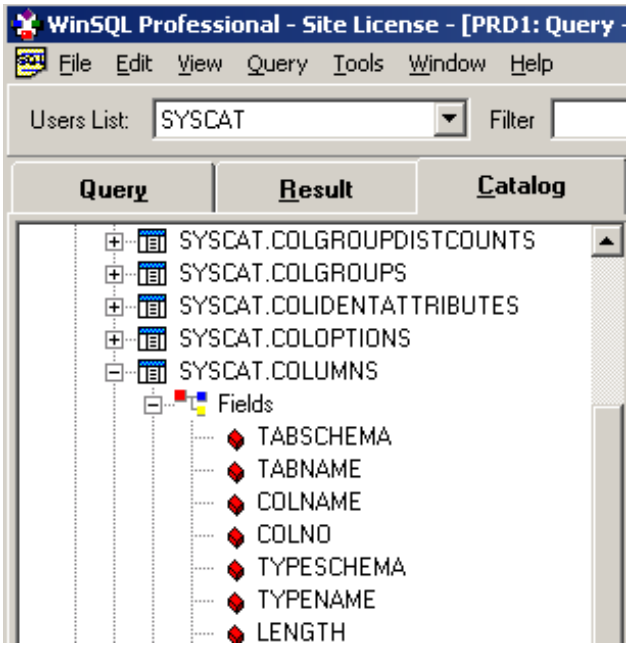
Teradata example from DBC.COLUMNS. DBC is the database name where Teradata dictionary data is found.



Querying **Teradata** dictionary data:

```
select tablename, columnname, columnformat, columntype
  from dbc.columns
 where databasename = 'your_db'
   and tablename    = 'your_table'
 order by columntype, columnname
   ;
```

DB2 example from SYSCAT.COLUMNS. SYSCAT is the schema where DB2 dictionary data is found:


WinSQL Professional - Site License - [PRD1: Query
File Edit View Query Tools Window Help
Users List: SYSCAT
Filter
Query
Result
Catalog
SYSCAT.COLGROUPDISTCOUNTS
SYSCAT.COLGROUPS
SYSCAT.COLIDENTATTRIBUTES
SYSCAT.COLOPTIONS
SYSCAT.COLUMNS
Fields
TABSCHEMA
TABNAME
COLNAME
COLNO
TYPESCHEMA
TYPENAME
LENGTH

Querying **DB2** dictionary data:

```
select tabschema, tabname, colname, typename
  from syscat.columns
 where tabschema = 'DROOGH2'
   and tabname   = 'QC_TEST'
 order by typename, colname
 ;
```

DB2 Dictionary Results:

```
TABSCHEMA TABNAME  COLNAME                  TYPENAME
--------- -------- ------------------------ --------
DROOGH2   QC_TEST  ACCT_ID                  BIGINT
DROOGH2   QC_TEST  CTD_CREDIT_AM            DECIMAL
DROOGH2   QC_TEST  CTD_DEBIT_AM             DECIMAL
DROOGH2   QC_TEST  DISPUT_AM                DECIMAL
DROOGH2   QC_TEST  CTD_CREDIT_CT            INTEGER
DROOGH2   QC_TEST  CTD_DEBIT_CT             INTEGER
DROOGH2   QC_TEST  ACCT_FAMILY_CD           SMALLINT
DROOGH2   QC_TEST  ACCT_SUBFAM_CD           SMALLINT
DROOGH2   QC_TEST  ACCT_TYPE_ID             SMALLINT
DROOGH2   QC_TEST  APPL_SUFFIX_NO           SMALLINT
DROOGH2   QC_TEST  CLIENT_PRODCT_CD         SMALLINT
DROOGH2   QC_TEST  TBAL_CD                  SMALLINT
DROOGH2   QC_TEST  ACCT_TYPE_MN             VARCHAR
DROOGH2   QC_TEST  ACCT_TYPE_NA             VARCHAR
```

The problem is beginning to emerge…. Each RDBMS system (and SAS) has different dictionary table names and different column names within those dictionary tables. If the intent is to create a generalized, data-driven utility to QC data, *any* data, it appears as though a roadblock has been encountered before we've begun. But…, doesn't SAS solve every problem?! **J**

SAS/ACCESS PRODUCTS

SAS is able to access non-SAS back-end data stores providing the appropriate SAS/Access product is licensed, eg.

```
SAS/ACCESS Interface to DB2
SAS/ACCESS Interface to ORACLE
```

Licensing the SAS/Access software provides the "middle-ware" that allows SAS to communicate with the RDBMS, issue queries and commands to the data base, and receive query results from the data base. Often queries are issued via "pass through" methodology where an SQL query *written in the syntax of the database* is "passed through" a connection to the data base and the results returned to SAS.

SAS/ACCESS LIBNAME ENGINE

Alternatively, SAS/Access also provides a LIBNAME engine which allows database accessibility in a manner very similar to that used for native SAS data sets. When a SAS/Access library is established, the database tables may be created, queried and manipulated by standard SAS data steps and SAS procedure steps.

The SAS/Access libname statement often requires additional parameters to provide authentication credentials and options specific to the data base engine for addressability and efficiency, eg.

```
libname _db2 db2      database=test schema=droogh2;
libname _td  teradata database=data_base_name  user=userid pass=password;
```

SURFACING RDBMS METADATA USING SAS/ACCESS

Once a LIBNAME connection has been established to the data base system, the DB metadata may be accessed using standard SAS procedures such as PROC CONTENTS. In the example below, tracing options have been specified to provide additional information on the operations going on under the covers.

Once the DB2 connection is made, the DB2 table metadata is being accessed via PROC CONTENTS.

```
options sastrace=',,,d' sastraceloc=saslog nostsuffix;

libname _db2 db2 database=test schema=droogh2;

proc contents data = _db2.qc_test;
run;
```

Log results showing the SASTRACE output:

```
DB2: AUTOCOMMIT is NO for connection 0
516  options sastrace=',,,d' sastraceloc=saslog nostsuffix;
518  libname _db2 db2 database=test schema=droogh2;
NOTE: Libref _DB2 was successfully assigned as follows:
      Engine:        DB2
      Physical Name: test

DB2: AUTOCOMMIT turned ON for connection id 0
DB2_1: Prepared:
SELECT * FROM droogh2.QC_TEST FOR READ ONLY
DB2: COMMIT performed on connection 0.
520  proc contents data = _db2.qc_test;
521  run;
NOTE: PROCEDURE CONTENTS used :
```

Behind the scenes additional processing is taking place that is not reflected in the SAS log. Through the SAS/Access engine SAS is making a series of calls to the back-end database to request the information required for PROC CONTENTS

- SQLNumResultCols — number of columns in table
- SQLDescribeCol — column name, type, length etc.
- SQLColAttribute — type specific column attributes

The end result is CONTENTS output that is quite familiar though the results are a little more sparse than is typically returned for a SAS data set :

```
The CONTENTS Procedure

Data Set Name        _DB2.QC_TEST   Observations          .
Member Type          DATA           Variables             14
Engine               DB2            Indexes               0
Created              .              Observation Length    0
Last Modified        .              Deleted Observations  0
Protection                          Compressed            NO
Data Set Type                       Sorted                NO
```

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Informat	Label
12	ACCT_FAMILY_CD	Num	8	11.	11.	ACCT_FAMILY_CD
1	ACCT_ID	Num	8	20.	20.	ACCT_ID
13	ACCT_SUBFAM_CD	Num	8	11.	11.	ACCT_SUBFAM_CD
10	ACCT_TYPE_ID	Num	8	11.	11.	ACCT_TYPE_ID
11	ACCT_TYPE_MN	Char	15	$15.	$15.	ACCT_TYPE_MN
14	ACCT_TYPE_NA	Char	23	$23.	$23.	ACCT_TYPE_NA
3	APPL_SUFFIX_NO	Num	8	11.	11.	APPL_SUFFIX_NO
2	CLIENT_PRODCT_CD	Num	8	11.	11.	CLIENT_PRODCT_CD
9	CTD_CREDIT_AM	Num	8	15.2	15.2	CTD_CREDIT_AM
8	CTD_CREDIT_CT	Num	8	11.	11.	CTD_CREDIT_CT
7	CTD_DEBIT_AM	Num	8	15.2	15.2	CTD_DEBIT_AM
6	CTD_DEBIT_CT	Num	8	11.	11.	CTD_DEBIT_CT
5	DISPUT_AM	Num	8	15.2	15.2	DISPUT_AM
4	TBAL_CD	Num	8	11.	11.	TBAL_CD

Despite the differences, it's apparent that the SAS/Access engine is returning table and column metadata for database systems in exactly the same way it's provided for SAS datasets. A generalized approach to surfacing metadata, regardless of the source of the data is possible.

QC YOUR DATA – A GENERALIZED APPROACH

Given a library reference pointing to the data store, whether it be SAS, DB2, Teradata, Oracle etc. it's possible to generate column metadata with enough information to begin to make sensible determinations for the initial QC exercise. While frequency distributions on many character fields may be helpful, it certainly won't be on a field containing customer name. Numeric analysis including min, max and mean will suffice for most numeric fields, but not for discrete numeric values that you might find in a column containing numeric codes.

The utility macro presented in the rest of this paper provides the methodology and options to drive the QC process. Let's walk through the highlights of the **%qc_db_data** macro.

MACRO HELP TEXT

Macros are great, especially if usage documentation is available. **J** Utility macros of this nature benefit from the inclusion of a *positional* parameter that will optionally generate "help" text in the log outlining the purpose of the macro and the parameters it expects, ie. **%qc_db_data(?)**. The resulting log text provides the macro documentation.

```
================================================================================================
%qc_db_data( help, lib=, table=, drop_columns=, keep_columns=, by_vars=, where=, freq_limit =
100)

QC / analyze the RDBMS table specified, creating frequency distributions or min, max, mean,
stddev and sum depending on the column type and granularity of the data in the table.

Parms:
   help             any value in the sole positional parameter provides this help text
   lib              SAS libref via RDBMS engine for schema that contains &table
   table            RDBMS table to be analyzed, MUST be sorted by &by_vars ( if specified )
   drop_columns     comma-delimited, single-quoted column names to be IGNORED in analysis,
                    - must use %str('col1','col2') when specifying multiple column names
                    - always specify 'acct_id', 'cust_id' type fields in this parm
   keep_columns     comma-delimited, single-quoted column names to be considered for analysis,
                    - must use %str('col1','col2') when specifying multiple column names
   by_vars          comma-delimited, single-quoted column names for BY groups
                    - must use %str('col1','col2') when specifying multiple column names
   where            WHERE clause to apply to input &schema..&table to focus analysis
   freq_limit       upper limit of number of distinct values used to decide which vars generate
```

```
                    frequency distributions, default is 100 distinct values
                    - all columns with <= &freq_limit distinct values will generate freq dist
                    - num columns with >  &freq_limit distinct values will generate num analysis

Macro logic outlined below:

  1. Derive table columns using PROC CONTENTS data=&lib..&table, incorporate &drop_column
     and &keep_column criteria
  2. count distinct values for all selected fields
  3. numeric fields where count of distinct values > &freq_limit, create min/max/stddev/sum stats
  4. run frequency distribution on any fields that have <= &freq_limit distinct values
  5. if &by_vars are specified, all stats will be created with the BY groups specified
  6. create datasets of final results in remwork._qc_continuous_data and
     remwork._qc_categorical_data

Sample Invocation:

libname rdbms <RDBMS engine> <RDBMS connection particulars>;

%qc_db_data(lib             = rdbms,
            table           = qc_test,
            drop_columns    = %str('acct_id'),
            by_vars         = %str('acct_type_na'),
            where           = %str(acct_type_na like 'SAV%'),
            freq_limit      = 50
            )
```

INVOKING THE MACRO

```
libname _db2 db2       database=test schema=droogh2;

%qc_db_data(
   lib          = _db2,
   table        = qc_test,
   drop_columns = %str('acct_id'),
   by_vars      = %str('acct_type_na'),
   where        = %str(acct_type_na like '%Visa%'),
   freq_limit   = 100
        );
```

As outlined in the macro generated help text, a library reference must be established before the macro is invoked. Since the library reference is established *outside* the macro the user has full control in defining the data source. The data can reside anywhere that is addressable via the LIBNAME statement. In the example above, the data to be examined resides in a DB2 table, hence the DB2 engine specification in the LIBNAME statement.

The macro parameters define the QC test particulars:

- lib= the data resides in the libref _db2
- table= the data table is qc_test
- drop_columns= acct_id is a numeric column, but since we're not interested in analyzing a table key, it is being dropped from the analysis
- by_vars= the results are required for each acct_type_na value, hence the by_vars parameter
- where= not all accounts are to be analyzed, only those with the word "Visa" in the account type name
- freq_limit= if less than 100 distinct values are found in a column, a frequency distribution will be generated, otherwise analysis will be performed on numeric columns (character columns with more than 100 distinct values will be ignored)

MACRO FUNCTIONALITY

Since the reader of this paper is a SAS programmer it's not necessary to walk through the entire macro (available in its entirety in the appendix). However, some important highlights of the macro functionality will be discussed. Explanatory comments follow each code segment.

```
proc contents data = &lib..&table
              out = _qc_db_columns_all
                           ( keep =   name type formatl
                           rename = ( name = colname )  ) noprint;
run;
```

Utilizing the availability of data table metadata provided by the LIBNAME (no matter where the data resides), use PROC CONTENTS to extract the columnar metadata, storing it in a SAS dataset, keeping only the data items required.

```
data _qc_db_columns;
 set _qc_db_columns_all;

 %if &drop_columns > %then %do;
    if colname not in ( %upcase(&drop_columns) );
    %end;

 %if &keep_columns > %then %do;
    if colname in ( %upcase(&keep_columns) );
    %end;

 if type = 1 then coltype = 'N'; else coltype = 'C';

 drop type;
run;
```

Since not all the variables in the selected table are good candidates for analysis, macro parameters allow variables to be specified in the KEEP and DROP parameters. In practice, it really only makes sense to specify one or the other. Based on the &keep_columns and &drop_columns parameter values, subset the table variables.

```
/*
    Create the count(distinct x) as x phrases.  The
    results of these will determine whether we do
    freq distribution on the variables
*/

select

'count (distinct(' || trim(colname) ||
    ')) as ' || trim(column_name)

 into :_qc_count_distinct separated by ','

 from _qc_db_columns
```

Continuing with the data-driven approach, programmatically generate "count distinct" SQL clauses from the table metadata. These counts will be used to decide if frequency distributions ought to be produced for each variable.

```
/*
    Count distinct values of each variable,
    these counts used to decide if
    min/max/etc.. or freqs to be done
*/
 create table _qc_count_distinct as
    select &_qc_count_distinct
      from &lib..&table
    %if &where ne %then %do;
       where &where      %end;   ;
```

Using the “count distinct” clauses built in the previous step, execute the counts against the source table, creating a table of the distinct counts by variable. Note the WHERE clause is created only if the &where parameter was specified when the macro was invoked. The result below is then transposed into a table called _qc_count_distinct_xpose to make the columns available as rows.

VIEWTABLE: Remwork._qc_count_distinct_x

	colname	cnt
1	ACCT_FAMILY_CD	1
2	ACCT_SUBFAM_CD	3
3	ACCT_TYPE_ID	10
4	ACCT_TYPE_MN	10
5	APPL_SUFFIX_NO	1
6	CLIENT_PRODCT_CD	11
7	CTD_CREDIT_AM	1979
8	CTD_CREDIT_CT	14
9	CTD_DEBIT_AM	19950
10	CTD_DEBIT_CT	65
11	DISPUT_AM	1
12	TBAL_CD	9

```
/* Numeric columns will be run through proc summary */

 select d.colname
   into :numeric_cols separated by ' '

   from _qc_db_columns          d,
        _qc_count_distinct_xposec

  where d.colname   = c.colname
    and d.coltype   = 'N'
    and c.cnt       > &freq_limit
;

 %let numeric_fld_cnt = &sqlobs;
```

The columns eligible for numeric analysis are those that are typed numeric and have *more* distinct values than the cutoff for frequency distribution. After the SELECT executes, note that the &numeric_fld_cnt macro variable will contain the number of variables requiring numeric analysis.

```
/*
     Any column with < &freq_limit distinct values is freqqed.
     This means that some character columns will have no analysis
     performed on them, eg. name fields.
*/

select d.colname, d.colname
  into :char_col1 - :char_col&sysmaxlong ,
       :char_cols separated by ' '

  from _qc_db_columns          d,
       _qc_count_distinct_xposec

 where d.colname   = c.colname
   and c.cnt    <= &freq_limit ;

%let char_fld_cnt  = &sqlobs;
```

Any column, character or numeric, that has less distinct values than &freq_limit will be subject to frequency distribution.

```
proc summary data = &lib..&table ( keep = &numeric_cols &by_vars_stmt )
              nway missing ;

 %if &where ne %then %do;
    where &where;
 %end;

 var &numeric_cols;

 %if &by_vars_stmt ne %then %do;
    by &by_vars_stmt notsorted;  *  RDBMS does not necessarily
                                    return rows in correct order
                                    for mixed-case character
                                    columns;
 %end;

 output out = _qc_metrics_num_n      ( drop = _: )  n= ;
 output out = _qc_metrics_num_min    ( drop = _: )  min= ;
 output out = _qc_metrics_num_max    ( drop = _: )  max= ;
 output out = _qc_metrics_num_mean   ( drop = _: )  mean= ;
 output out = _qc_metrics_num_stddev ( drop = _: )  stddev=;
 output out = _qc_metrics_num_sum    ( drop = _: )  sum= ;
run;
```

The table columns identified as candidates for numeric analysis are processed by PROC SUMMARY. Note that the WHERE and BY statements will only be included if required. As per the macro documentation, if BY variables are being used, the incoming data MUST be sorted ! Since some databases ignore the case of column values when sorting, NOTSORTED is specified on the BY statement in the event SAS and the back-end data store use different collation schemes.

The separate output datasets created for each measure will be transposed and merged back together towards the end of the process. Note as well the KEEP data set option where only the required variables are returned from the data source. This makes a big difference in execution time when accessing large tables from DB systems.

```
proc freq data = &lib..&table ( keep = &char_cols &by_vars_stmt );

 %if &where ne %then %do;
    where &where;
 %end;
 * RDBMS sort order for mixed-case character columns differs;
 %if &by_vars_stmt ne %then %do;
    by &by_vars_stmt notsorted;
 %end;

 %do i = 1 %to &char_fld_cnt;
    tables &&char_col&i / missing
      out = &&char_col&i ( rename = ( &&char_col&i = value )) ;
 %end;
run;
```

Run a PROC FREQ for the categorical character and numeric variables, specifying each variable in a separate TABLE statement in order to save the frequency distributions in SAS datasets.

After a series of transposes and merges, the finished results are displayed in two datasets, one for categorical variables, the other for continuous.

VIEWTABLE: Remwork._qc_categorical_data

	colname	value	ACCT_TYPE_NA	COUNT	PERCENT
56	CLIENT_PRODCT_CD	10223	Mellow Yellow Visa Card	6246	22.839799612
57	CLIENT_PRODCT_CD	10224	Mellow Yellow Visa Card	21101	77.160200388
58	CLIENT_PRODCT_CD	12271	Platinum Visa Card	255	100
59	CLIENT_PRODCT_CD	10226	Student Visa Card	21557	100
60	CLIENT_PRODCT_CD	10225	US Visa Card	1624	100
61	CLIENT_PRODCT_CD	10227	Visa Check Card run	1235	100
62	CTD_CREDIT_CT	0	Dividend Visa Card	17267	97.719298246
63	CTD_CREDIT_CT	1	Dividend Visa Card	308	1.743067345
64	CTD_CREDIT_CT	2	Dividend Visa Card	61	0.345217883

VIEWTABLE: Remwork._qc_continuous_data

	ACCT_TYPE_NA	Column Name	n	min	max	mean	stddev	sum
1	Dividend Visa Card	CTD_CREDIT_AM	17,670	0.00	2,513.36	2.86	42.89	50,562.80
2	Dividend Visa Card	CTD_DEBIT_AM	17,670	0.00	10,500.00	84.04	372.10	1,485,008.04
3	Gold Plus Visa Card	CTD_CREDIT_AM	6,044	0.00	11,992.72	11.25	190.59	68,000.08
4	Gold Plus Visa Card	CTD_DEBIT_AM	6,044	0.00	84,070.80	286.87	1,605.78	1,733,842.93
5	Gold Premium Visa Card	CTD_CREDIT_AM	9,523	0.00	4,569.57	5.02	78.28	47,838.08
6	Gold Premium Visa Card	CTD_DEBIT_AM	9,523	0.00	19,400.00	148.88	613.30	1,417,760.22
7	Gold Visa Card	CTD_CREDIT_AM	4,805	0.00	2,437.03	8.89	76.05	42,697.52
8	Gold Visa Card	CTD_DEBIT_AM	4,805	0.00	37,202.36	252.84	939.28	1,214,915.85
9	Green Power Visa Card	CTD_CREDIT_AM	31,847	0.00	20,000.00	2.33	115.80	74,295.01
10	Green Power Visa Card	CTD_DEBIT_AM	31,847	0.00	20,000.00	68.19	374.47	2,171,704.34

Note the presence of the BY variable in each dataset. Results are presented within each BY variable value. The WORK datasets may be saved to permanent libraries or output into presentation quality reports using ODS. The usefulness of this macro could easily be extended for periodic QC procedures on the same type of data by keeping figures for each period and comparing period over period values.

CONCLUSION

Flexible, maintenance-free, data-driven code is made possible by leveraging metadata. The SAS/Access engine provides seamless access to back-end data and allows standard SAS PROCs to be run against database tables providing rudimentary data analysis that can assist in rapid data discovery.

Please check my website below for the latest version of the **%qc_db_tables** macro as improvements will be made!

REFERENCES

SAS/Access 9.2 For Relational Databases Reference, Third Edition,
http://support.sas.com/documentation/cdl/en/acreldb/63283/HTML/default/viewer.htm#/documentation/cdl/en/acreldb/63283/HTML/default/titlepage.htm

CONTACT INFORMATION

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

Harry Droogendyk
Stratia Consulting Inc.
905-296-3595
www.stratia.ca

```
/*****************************************************************************************************************************

   Author:        Harry Droogendyk             harry@stratia.ca

   Date:       2010-01-28

   SAS macro to provide analysis of RDBMS table contents.  See the macro document for more information.  To display
   macro documentation, invoke the macro with a single positional parameter:

      %qc_db_data(?)

-----------------------------------------------------------------------------------------------------------------------------
Modifications:
-----------------------------------------------------------------------------------------------------------------------------
Date        Who               What
-----------------------------------------------------------------------------------------------------------------------------


*****************************************************************************************************************************/

%macro qc_db_data(help,
      lib      =,
      table    =,
      drop_columns   =,
      keep_columns   =,
      by_vars     =,
      where    =,
      freq_limit  = 100
      );

   %if %length(&help) > 0 %then %do;
      %put %nrstr();
      %put %nrstr(=============================================================================================);
      %put %nrstr(%qc_db_data( help, lib=, table=, drop_columns=, keep_columns=, by_vars=, where=, freq_limit = 100 ) );
      %put %nrstr();
      %put %nrstr(QC / analyze the RDBMS table specified, creating frequency distributions or min, max, mean,  );
      %put %nrstr(stddev and sum depending on the column type and granularity of the data in the table.  );
      %put %nrstr();
      %put %nrstr(Parms:);
      %put %nrstr(   help           any value in the sole positional parameter provides this help text );
      %put %nrstr(   lib            SAS libref via RDBMS engine for schema that contains &table );
      %put %nrstr(   table          RDBMS table to be analyzed, MUST be sorted by &by_vars ( if specified ) );
      %put %nrstr(   drop_columns   comma-delimited, single-quoted column names to be IGNORED in analysis,);
      %put %nrstr(                  - must use %str('col1','col2') when specifying multiple column names);
      %put %nrstr(                  - always specify 'acct_id', 'cust_id' type fields in this parm);
      %put %nrstr(   keep_columns   comma-delimited, single-quoted column names to be considered for analysis,);
      %put %nrstr(                  - must use %str('col1','col2') when specifying multiple column names);
      %put %nrstr(   by_vars        comma-delimited, single-quoted column names for BY groups);
      %put %nrstr(                  - must use %str('col1','col2') when specifying multiple column names);
      %put %nrstr(   where          WHERE clause to apply to input &schema..&table to focus analysis);
      %put %nrstr(   freq_limit     upper limit of number of distinct values used to decide which vars generate );
      %put %nrstr(                  frequence distributions, default is 100 distinct values);
      %put %nrstr(                  - all columns with <= &freq_limit distinct values will generate freq dist);
      %put %nrstr(                  - num columns with >  &freq_limit distinct values will generate num analysis);
```

```
   %put %nrstr();
   %put %nrstr(Macro logic outlined below:);
   %put %nrstr();
   %put %nrstr(  1. Derive table columns using PROC CONTENTS data=&lib..&table, incorporate &drop_column  );
   %put %nrstr(     and &keep_column criteria);
   %put %nrstr(  2. count distinct values for all selected fields);
   %put %nrstr(  3. numeric fields where count of distinct values > &freq_limit, create min/max/stddev/sum stats);
   %put %nrstr(  4. run frequency distribution on any fields that have <= &freq_limit distinct values);
   %put %nrstr(  5. if &by_vars are specified, all stats will be created with the BY groups specified);
   %put %nrstr(  6. create datasets of final results in remwork._qc_continuous_data and );
   %put %nrstr(     remwork._qc_categorical_data);
   %put %nrstr();
   %put %nrstr(Sample Invocation:);
   %put %nrstr();
   %put %nrstr(libname rdbms <RDBMS engine> <RDBMS connection particulars>;);
   %put %nrstr();
   %put %nrstr(%qc_db_data%(lib           = rdbms,);
   %put %nrstr(             table         = qc_test,);
   %put %nrstr(             drop_columns  = %str('acct_id'),);
   %put %nrstr(             by_vars       = %str('acct_type_na'),);
   %put %nrstr(             where         = %str(acct_type_na like 'SAV%'),);
   %put %nrstr(             freq_limit    = 50);
   %put %nrstr(      ));
   %put %nrstr(=============================================================================================);
   %put %nrstr();

   %return;
%end;

%local   by_vars_stmt sample  ;

/*  Clean up results datasets before we begin  */

proc datasets lib = work nodetails nolist;
   delete _qc:    / mtype = data;
   run;
quit;

/*  If BY vars have been specified, clean up the quotes so we can use the variable names in a BY statement  */

%if &by_vars ne %then %do;
   %if &drop_columns ne %then
      %let drop_columns = &drop_columns, &by_vars;
   %else
      %let drop_columns = &by_vars;
   %let by_vars_stmt = %sysfunc(compress(&by_vars,%str(%')));  /*  'For use in BY statements  */
%end;

/*
   Identify character / numeric fields in the table.  If &keep_columns / &drop_columns
   have been specified, we'll use that to define the columns we care deeply about.  Creating
   a SAS table of the RDBMS table columns we're interested in.

   We're using the format as a proxy for length since SAS will return 8 for all numeric fields,
```

```
   we want the actual format.
*/

proc contents data = &lib..&table
              out = _qc_db_columns_all ( keep = name type formatl
                                         rename = ( name = colname )
                                        )
              noprint;
run;

data _qc_db_columns;
   set _qc_db_columns_all;
   %if &drop_columns > %then %do;
      if colname not in ( %upcase(&drop_columns) );
      %end;

   %if &keep_columns > %then %do;
      if colname in ( %upcase(&keep_columns) );
      %end;

   if type = 1 then coltype = 'N'; else coltype = 'C';

   drop type;
run;

proc sort data = _qc_db_columns;
           by   colname;
run;

/*  Need the maximum variable length for a later step  */

proc sql;
   select max(formatl)
     into :max_length
     from _qc_db_columns
   ;

   %let max_length = &max_length;

   /*
      Create the count(distinct x) as x phrases.  The results of
      these will determine whether we do freq dist on the variables
   */

   select 'count (distinct(' || trim(colname) || ')) as ' || trim(colname)
     into :_qc_count_distinct separated by ','
     from _qc_db_columns
   ;

   /*  Count distinct values of each variable, these counts used to decide if min/max/etc.. or freqs to be done  */

   create table _qc_count_distinct as
      select &_qc_count_distinct
       from &lib..&table
```

```
      %if &where ne %then %do;
         where &where
      %end;
   ;
quit;

proc transpose data = _qc_count_distinct
               out = _qc_count_distinct_xpose ( rename = ( _name_ = colname col1 = cnt )) ;
   var _numeric_;
run;

/*
   If the count distinct has found < &freq_limit distinct values, treat the variable
   as a categorical variable, even if it is numeric
*/

%let numeric_fld_cnt    = 0;
%let char_fld_cnt    = 0;

proc sql;

   /*  Numeric columns will be run through proc summary  */

   select d.colname
     into :numeric_cols separated by ' '

     from _qc_db_columns         d,
         _qc_count_distinct_xpose   c

    where d.colname  = c.colname
      and d.coltype  = 'N'
      and c.cnt      > &freq_limit
   ;

   %let numeric_fld_cnt = &sqlobs;

   /*
      Any column with < &freq_limit distinct values is freqqed.  This means that some
      character columns will have no analysis performed on them, eg. name fields.
   */

   select d.colname, d.colname
     into :char_col1 - :char_col&sysmaxlong
       , :char_cols  separated by ' '

     from _qc_db_columns         d,
         _qc_count_distinct_xpose   c

    where d.colname  = c.colname
      and c.cnt      <= &freq_limit
   ;

   %let char_fld_cnt = &sqlobs;
```

```
quit;

%if &numeric_fld_cnt = 0 and &char_fld_cnt = 0 %then %do;
   %put;
   %put No numeric or character fields on the table, that IS a mystery!!  Aborting;
   %put;
   %return;
%end;

/*
   Generate numeric analysis.  If the BY vvars are in play, the value of COUNT will be equal
   to the number of rows for each particular BY slice.
*/

%if &numeric_fld_cnt ne 0 %then %do;

   proc summary data = &lib..&table ( keep = &numeric_cols &by_vars_stmt ) nway missing ;

      %if &where ne %then %do;
         where &where;
      %end;

      var &numeric_cols;

      %if &by_vars_stmt ne %then %do;
         by &by_vars_stmt notsorted;   * DB2 does return rows in correct order for mixed-case character columns;
      %end;

      output out = _qc_metrics_num_n      ( drop = _: )  n= ;
      output out = _qc_metrics_num_min    ( drop = _: )  min= ;
      output out = _qc_metrics_num_max    ( drop = _: )  max= ;
      output out = _qc_metrics_num_mean   ( drop = _: )  mean= ;
      output out = _qc_metrics_num_stddev ( drop = _: )  stddev=;
      output out = _qc_metrics_num_sum    ( drop = _: )  sum= ;
   run;

   /*
      Keeping _type_ around until now since I was unsure whether we'd want NWAY
      or individual summary points for each BY variable
   */

   proc transpose data = _qc_metrics_num_n
                  out = _qc_metrics_num_n_xpose   ( drop = _label_
                                                    rename = ( _name_ = colname col1 = n )) ;
      var _numeric_;
      %if &by_vars_stmt ne %then %do;
         by &by_vars_stmt notsorted;
      %end;
   run;
   proc transpose data = _qc_metrics_num_min
                  out = _qc_metrics_num_min_xpose ( drop = _label_
                                                    rename = ( _name_ = colname col1 = min )) ;
      var _numeric_;
      %if &by_vars_stmt ne %then %do;
```

```
      by &by_vars_stmt notsorted;
   %end;
run;
proc transpose data = _qc_metrics_num_max
               out = _qc_metrics_num_max_xpose ( drop = _label_
                                                 rename = ( _name_ = colname col1 = max )) ;
   var _numeric_;
   %if &by_vars_stmt ne %then %do;
      by &by_vars_stmt notsorted;
   %end;
run;
proc transpose data = _qc_metrics_num_mean
               out = _qc_metrics_num_mean_xpose ( drop = _label_
                                                  rename = ( _name_ = colname col1 = mean )) ;
   var _numeric_;
   %if &by_vars_stmt ne %then %do;
      by &by_vars_stmt notsorted;
   %end;
run;
proc transpose data = _qc_metrics_num_stddev
               out = _qc_metrics_num_stddev_xpose ( drop = _label_
                                                    rename = ( _name_ = colname col1 = stddev )) ;
   var _numeric_;
   %if &by_vars_stmt ne %then %do;
      by &by_vars_stmt notsorted;
   %end;
run;
proc transpose data = _qc_metrics_num_sum
               out = _qc_metrics_num_sum_xpose ( drop = _label_
                                                 rename = ( _name_ = colname col1 = sum )) ;
   var _numeric_;
   %if &by_vars_stmt ne %then %do;
      by &by_vars_stmt notsorted;
   %end;
run;

proc sort      data = _qc_metrics_num_n_xpose;        by &by_vars_stmt colname; run;
proc sort      data = _qc_metrics_num_min_xpose;      by &by_vars_stmt colname; run;
proc sort      data = _qc_metrics_num_max_xpose;      by &by_vars_stmt colname; run;
proc sort      data = _qc_metrics_num_mean_xpose;     by &by_vars_stmt colname; run;
proc sort      data = _qc_metrics_num_stddev_xpose;   by &by_vars_stmt colname; run;
proc sort      data = _qc_metrics_num_sum_xpose;      by &by_vars_stmt colname; run;

data _qc_continuous_data;
   merge _qc_metrics_num_n_xpose
         _qc_metrics_num_min_xpose
         _qc_metrics_num_max_xpose
         _qc_metrics_num_mean_xpose
         _qc_metrics_num_stddev_xpose
         _qc_metrics_num_sum_xpose;
      by
      %if &by_vars_stmt ne %then %do;
         &by_vars_stmt
      %end;
```

```
         colname ;

      format   min max mean stddev sum comma24.2
               n  comma15.
            ;

      label    colname     = 'Column Name' ;

   run;

%end;

/*  Loop over char fields ( or numeric vars with < &freq_limit granularity ) running a FREQ on each  */

%if &char_fld_cnt ne 0 %then %do;

   proc freq data = &lib..&table ( keep = &char_cols &by_vars_stmt );

      %if &where ne %then %do;
         where &where;
      %end;

      %if &by_vars_stmt ne %then %do;
         by &by_vars_stmt notsorted;      * DB2 does not return rows in correct order  ;
      %end;

      %do i = 1 %to &char_fld_cnt;
         tables &&char_col&i / missing out = _qc_freq_&i ( rename = ( &&char_col&i = value )) ;
      %end;
   run;

   %if &max_length < 32 %then
      %let best = best&max_length;
   %else
      %let best = best32;

   %do i = 1 %to &char_fld_cnt;
      data _qc / view = _qc;
         length colname    $32
                value      $&max_length
         ;

         retain colname "&&char_col&i";

         set _qc_freq_&i ( rename = ( value = _val ));

         /*
            We can't mix numeric / character fields, so convert all numeric to character.  We have to use
            PUTC/PUTN to "hide" the format from the compiler otherwise it kaks when it sees what it thinks
            is a numeric format for character fields, ie. at compile time it doesn't consider the conditional
            stmt that would prevent such a thing from happening.
         */

         if vtype (_val) = 'N' then do;
```

```
            *put "&&char_col&i - in num";
            _fmt  = "&best";          * if max_length is less than 16, we take the chance of losing digits !!!!   ;
            if missing(_val) then
               value = 'null';
            else
               value = putn(_val,_fmt);
         end; else do;
            *put "&&char_col&i - in char";
            _fmt  = "$&max_length";
            value = putc(_val,_fmt);
         end;

         drop _: ;
      run;

      proc append base = _qc_categorical_data
               data = _qc              force;
      run;
   %end;

%end;

%let source = %sysfunc(getoption(source));
options nosource;

%put;
%put ------------------------------------------------------------------------------------------;
%put %nrstr(%qc_db_data has completed, please check the log for any errors.) ;
%put %nrstr(Successful completition will result in the creation of two datasets:) ;
%put %nrstr(    _qc_categorical_data - categorical variable value distributions) ;
%put %nrstr(    _qc_continuous_data  - continuous data analysis, eg. min, max etc...) ;
%put;
%put %nrstr(Data from these two tables can be viewed from the SAS Explorer, exported to Excel) ;
%put %nrstr(or printed ( perhaps with appropriate ODS wrapper statements ) to create output);
%put ------------------------------------------------------------------------------------------;
%put ;

options &source;

%mend qc_db_data;
```