PhUSE 2010

Paper CC04

Updating an MS SQL database from SAS

Jim Groeneveld, OCS Biometric Support, ‘s Hertogenbosch, Netherlands

SUMMARY

Updating an MS SQL database table from SAS® PROC SQL without write access to the whole database, but to just one particular, already existing table is possible, but more complicated than just creating a new table from scratch. The MS SQL table does not need to have a structure already, possibly existing content and the structure can even be removed and a **new structure** can be built and **new data** can be transferred. This is possible via the SAS/ACCESS LIBNAME statement and by using the Pass-Through Facility.

A drawback is that, contrary to having SAS automatically take care of it when using the CREATE statement, the conversion of DATE, TIME and DATETIME fields into MS SQL datetime fields is not done automatically and should be explicitly coded. However, this can still largely be automated by looking at the defined FORMATs of the variables concerned.

This paper describes what would happen if one would have permission to **CREATE** an MS SQL table from PROC SQL, including the **automatic conversion** of DATE, TIME and DATETIME (and DOLLAR into MONEY) fields, based on their **date and time related formats**, via the SAS/ACCESS engine. But as a common SAS user should only have restricted access to an MS SQL database; the engine cannot be used to CREATE an MS SQL table. The export of data to update an MS SQL table should then be coded explicitly. Of course, when reading from an MS SQL database table via the ODBC engine into SAS, for which one has read permission, SAS provides automatic date and time conversions.

The paper discusses the code to explicitly convert date and time fields by way of their associated DATE, TIME and DATETIME **formats** (like SAS does). As MS SQL only knows DATETIME fields SAS DATE fields have to be multiplied by 86400. It is highly recommended to convert all combinations of DATE and TIME variables in SAS to DATETIME variables beforehand. SAS and MS SQL use the same offset for their DATETIME fields, so no further correction to that extent is necessary.

Furthermore the code to access an existing MS SQL table for which one has write permission is presented. At first the **existing content should be deleted** after which the **structure has to be removed**. In SAS, datasets without any variables are allowed, but in MS SQL a table should contain at least one column (variable). In order to delete the structure a dummy variable will be added to the MS SQL table initially after which all other variables are removed.

When **creating the new structure** for the MS SQL table, the new variables are written to the table after which the dummy variable is removed. All this can be done via the Pass-Through Facility. Finally **the new data** itself have to be **transferred** to the MS SQL table in the order that the columns have been declared. This is done using the LIBNAME statement with the ODBC engine. Basic code to do so, using macro variables with generated content, is:

```
LIBNAME MSSQLsrv ODBC DSN="SQL server" USER=username PWD=password;  * MS SQL lib ODBC engine;
PROC SQL;
  CONNECT TO ODBC (DSN="SQL server" USER=username PWD=password);    * Pass-Through Facility;
  EXEC (DELETE FROM &SQLtable) BY ODBC;                            * remove existing data;
  EXEC (ALTER TABLE &SQLtable ADD &Dummy FLOAT) BY ODBC;           * add a Dummy variable;
  EXEC (ALTER TABLE &SQLtable DROP COLUMN &SQLvars) BY ODBC;       * remove old structure;
  EXEC (ALTER TABLE &SQLtable ADD &SASvars) BY ODBC;               * define new structure;
  EXEC (ALTER TABLE &SQLtable DROP COLUMN &Dummy) BY ODBC;         * remove Dummy variable;
  DISCONNECT FROM ODBC;
  INSERT INTO &SQL_table SELECT * FROM &SAS_dataset;               * export data via ODBC;
QUIT;
```

Yet there is more preparation needed:

a. the **existing table's row structure** has to be obtained in order to be able to remove it (&MSSQLvars), via either SAS SQL CREATE or a data step;

b. the insertion of data in the last step must take place **in the same order** that the variables (columns) have been defined, it is just data, only values, no variables names or other information indicating which columns the values belong to..

INTRODUCTION

READING FROM AN MS SQL DATABASE

Transferring or copying an MS SQL table to a SAS dataset is very simple. First of all one should have a (once defined) SAS libname, specifying the ODBC engine, like:

```
LIBNAME MSSQLlib ODBC DSN=&DSN USER=&username PWD=&password;
```

in which &DSN = "SQL server" and a valid username and password have been given. Then there are three solutions:

1. a data step:

```
DATA SASlib.dataset;
   SET SQLlib.table;
RUN;
```

2. PROC DATASETS:

```
PROC DATASETS NOLIST; * (or outdated PROC COPY) ;
   COPY IN=SQLlib OUT=SASlib;
   SELECT table;
QUIT;
```

3. SAS PROC SQL:

```
PROC SQL;
   CREATE SASlib.dataset AS
   SELECT * FROM SQLlib.table;
QUIT;
```

All three alternatives create a new SAS dataset (or overwrite an existing one). Copying data from MS SQL to SAS needs read access on a whole or partial database or just a single (existing) table.

WRITING TO AN MS SQL DATABASE

Transferring a SAS dataset to an MS SQL table is similar. There are again three solutions, which are the same as when reading, these require to reverse the original and destination dataset/table or libname specifications. All three alternatives create a new table in the MS SQL database (or overwrite an existing one).

WRITE ACCESS DILLEMMA

It is quite evident that in this case SAS writes to the MS SQL database and needs write access rights to it, at least to one (or more) specific tables, whether existing or not. However, it is not possible to allow write permission to a non-existing table and even allowing an existing table to be overwritten/recreated needs creation rights. This implies that SAS would need to be allowed (with the appropriate username and password) to write anywhere in the database, to create, overwrite (and delete) tables to one's liking. Such overall access rights are not very safe nor desirable.

It is a security flaw if a user/SAS would be able to (over)write any table in the existing database. SAS should only have limited access to the database; it should only be allowed to create one (or more) new table(s). However, SAS should as well be able to overwrite its own previously created table(s), updating them with the latest data. But even in case SAS has write permission for a particular, existing table a SAS SQL CREATE command or a data step (with a LIBNAME pointing to the database table) will technically create a **new** table for which no access rights can be defined.

SOLUTION

In order to overcome this problem a method has been developed that allows SAS to write to an **existing** table, for which it has write permission. With this method SAS does not create or recreate a new table, but only modifies the content of the existing table. The method described here does not only permit to write (new) data to the existing structure (variables, columns) of the table, but it also allows to change its structure (variables) completely, just like when creating a new table.

The basic steps taken and discussed below in more detail are:

1. connecting to the database (with username and password);
2. emptying the table, removing all existing data, still leaving the structure;
3. removing the existing structure, the columns and their attributes;
4. defining the new structure (variables and attributes from the SAS dataset);
5. filling the table with data from the SAS dataset.

Finally a SAS macro **%SAStSQLt** is presented that makes it easy to recreate an existing MS SQL table from scratch, It only needs write permission for that specific, single, existing table and it transfers data by emptying the table first and refilling it subsequently. This may seem easy and straightforward but there are a lot of obstacles to be overcome as will become evident.

ELABORATION

In the following paragraphs each of the above steps is discussed in detail (sometimes divided into sub steps).

CONNECT TO MS SQL DATABASE

Connecting to the MS SQL database is not the most difficult part. It can easily be done using the ODBC engine in the LIBNAME statement and/or via the Pass-Through Facility. Examples of each of them used in this paper are:

```
LIBNAME MSSQLsrv ODBC DSN=”SQL server” USER=username PWD=password;  * MS SQL lib ODBC engine;
PROC SQL;
  CONNECT TO ODBC (DSN="SQL server" USER=username PWD=password);    * Pass-Through Facility;
```

EMPTY THE DATABASE TABLE'S CONTENTS

Removing all existing data from a database table while still leaving the structure intact can be done by the SQL Pass-Through EXEC statement:

```
  EXEC (DELETE FROM &SQLtable) BY ODBC;                            * remove existing data;
```

in which the SAS macro variable &SQLtable contains the name of the MS SQL table concerned. This result could as well be obtained using the equivalent SAS SQL statement (which does not use the SQL Pass-Through facility):

```
  DELETE FROM &SQL_table;  * (&SQL_table is SAS name of database table: MSSQLsrv.&SQLtable);
```

REMOVE THE DATABASE TABLE'S STRUCTURE

DROPping one or more columns from an MS SQL database table is very well possible, but it is not possible to DROP all columns at once. The reason for that is that, contrary to a SAS dataset or table, an MS SQL table should always have at least one column left; it may not be entirely empty.

In order to reach the desired goal, the removal of all existing columns, a dummy column is added to the MS SQL table's structure initially, after which all other named columns are being removed. The dummy column remains present for the time being. Adding the dummy column is done using the SQL Pass-Through facility:

```
  EXEC (ALTER TABLE &SQLtable ADD &Dummy FLOAT) BY ODBC;            * add a Dummy variable;
```

in which the SAS macro variable Dummy contains the name of the table's dummy column.

Subsequently all other columns can be removed via the SQL Pass-Through facility:

```
  EXEC (ALTER TABLE &SQLtable DROP COLUMN &SQLvars) BY ODBC;        * remove old structure;
```

Here the SAS macro variable &SQLvars contains a list of column names that exist in the database table. This list has been built beforehand by requesting it from the database table as in the code below. It could be built from the SAS SQL statement:

```
  SELECT Name INTO :SQLvars SEPARATED BY ' '
    FROM DICTIONARY.COLUMNS
    WHERE UPCASE(LIBNAME)='MSSQLSRV' AND UPCASE(MEMNAME)="&SQLtable";
```

Alternatively the list of columns (variables) could be derived from PROC CONTENTS, which in this case is done by using the SAS macro %Var_List (presented later in this paper) as:

```
%* Create existing variable list from MS SQL database table (to be removed);
%LET SQLvars = ; %* create the macro variable with empty contents;
%Var_List (DATA=&SQL_table, StoreVar=SQLvars, Delim=%STR(,)) /* (semicolon redundant) */
%PUT SQLvars=&SQLvars; %* feedback of existing columns in MS SQL table;
```

Adding and dropping database columns using SAS SQL code is apparently not possible and yielded the SAS error:

```
ERROR: The HEADER/VARIABLE UPDATE function is not supported by the ODBC engine
```

The SQL Pass-Through facility suffices here though. At this point the database table contains only the column Dummy.

DEFINE NEW DATABASE TABLE'S STRUCTURE

New columns have to be added to the database table. As explained above this apparently cannot be done using SAS SQL because of the generated error. That is why it should and can be done via the SQL Pass-Through facility by:

```
  EXEC (ALTER TABLE &SQLtable ADD &SASvars) BY ODBC;                * define new structure;
```

Like with any command passed to the server's SQL processor it has to be written with the syntax of MS SQL, occasionally somewhat different from the SAS SQL dialect. That means that in this case the generated command must be syntactically correct MS SQL language. Every column name should be followed by its type (VARCHAR(..), FLOAT, DATETIME or whatever) and the list of declarations must be delimited by commas. Note that MS SQL has more types of columns (variables) than SAS. On the other hand this version does not have separate DATA and TIME types, it has just one DATETIME type.

Furthermore, when SAS CREATEs an MS SQL table it looks at the variable's formats to see whether certain (SAS-numeric) variables are intended to be DATE-, TIME- or DATETIME variables. From that SAS automatically determines which type a column in MS SQL should have. Yet, in this case the CREATE command cannot be used as indicated before, because of security reasons. That is why the column and type declaration for MS SQL has to be coded explicitly while adding columns

and the types of the numeric SAS variables have to be determined from their associated formats, if any. That means that all possible Date, Time, DateTime and currency formats have to be checked explicitly and individually.

The preceding code below generates the MS SQL syntax of the column list declaration including the column types (into the macro variable &SASvars) and takes care of Date and Time type changes. The code involves several macro variables with lists of SAS formats, a macro and its call, %Var_List, that determines and processes the variable's types and formats:

```
%*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~;
%* To interpret variables as representing special types like DATE, TIME and DATETIME ;
%* all concerning formats have to be checked in order to interpret variables correct.;
%*_________________________________________________________________________________;

%* Format lists;
%LET DtTm_Fmt = 'DATETIME' 'DATEAMPM';
%LET Date_Fmt = 'DATE' 'DAY' 'DDMMYY' 'DOWNAM' 'EURDFD'
      'EURDFM' 'EURDFW' 'HDATE' 'HEBDAT' 'JULDAY' 'JULIAN' 'MINGUO' 'MMDDYY' 'MMYY'
      'MONNAM' 'MONTH' 'MONYY' 'NENGO' 'NLDATE' /*'NLDATM'*/ 'PDJULG' 'PDJULI' 'QTR'
       'WEEK' /*'WEEKDA'*/ 'WORDDA' 'YEAR' 'YYMM' 'YYMMDD' 'YYMON' 'YYQ';
%LET Time_Fmt = 'TIME' 'TOD' 'HHMM' 'HOUR' 'MMSS' /*'TIMEAMPM'*/;
%LET MoneyFmt = 'DOLLAR' ;
%* These lists may need adaptation with newer SAS versions;
%* Most macro variables are needed both inside and outside macro Var_List;

%* = = = = = = = =;
%* Macro Var_List ; %* Determination of variable type for MS SQL table ;
%MACRO Var_List (DATA=_LAST_, StoreVar=Var_List /* &Storevar should exist*/,
                Delim=%STR( ), Type=/*SAS/SQL*/);
  %LOCAL Contents;
  %LET Contents = ____List;
  PROC CONTENTS DATA=&Data OUT=&Contents (KEEP=NAME TYPE FORMAT VARNUM LENGTH) NOPRINT; RUN;
  PROC SORT DATA=&Contents; BY VarNum; RUN; * forcing variables in occurring order;
  DATA _NULL_;
    SET &Contents;
    LENGTH Attr $ 48;
    Attr = Name; * initialize with the variable (column) name;
    %IF (%UPCASE(&Type) EQ SAS) %THEN %DO;
      IF (Type EQ 2) THEN Attr = TRIM(Attr) || ' CHAR';
      ELSE IF (Type EQ 1) THEN Attr = TRIM(Attr) || ' NUM';
    %END;
    %ELSE %IF (%UPCASE(&Type) EQ SQL) %THEN %DO; %* deduce date and time variables;
      IF (Type EQ 2) THEN DO;
        IF (Length LE 8000) THEN Attr = TRIM(Attr) || ' VARCHAR(' ||
                                             TRIM(LEFT(PUT(Length,5.))) || ')';
        ELSE Attr = TRIM(Attr) || ' VARCHAR(MAX)';  * if allowed ; * at least 1024;
      END;
      ELSE IF (Type EQ 1) THEN DO;
        IF      (Format IN: (&DtTm_Fmt)) THEN Attr = TRIM(Attr) || ' DATETIME';
        ELSE IF (Format IN: (&Date_Fmt)) THEN Attr = TRIM(Attr) || ' DATETIME';
        ELSE IF (Format IN: (&Time_Fmt)) THEN Attr = TRIM(Attr) || ' DATETIME';
        ELSE IF (Format IN: (&MoneyFmt)) THEN Attr = TRIM(Attr) || ' MONEY';
        ELSE /* just numeric */               Attr = TRIM(Attr) || ' FLOAT';
      END;
    %END;
%*  else if &Type is neither SAS nor SQL then no attributes added to the names;
%* Build the column name+attr list in a single macro variable &StoreVar;
    CALL EXECUTE ('%LET &StoreVar = &&&StoreVar.&Delim.'||TRIM(Attr)||';');
  RUN;
%* remove initial delimiter &Delim;
  %LET &StoreVar = %SUBSTR(&&&StoreVar,2);
%MEND Var_List ;
%*= = = = = = =;

%* Macro call: Create variable list from SAS dataset;
%LET SASvars = ; %* create the macro variable with empty contents;
```

```
%Var_List (DATA=&Dataset, StoreVar=SASvars, Delim=%STR(,), Type=SQL) /* (no semicolon) */
%PUT SASvars=&SASvars; %* feedback of existing variables (and MS SQL attr) in SAS dataset;
```

With the information stored in the macro variable &SASvars it is possible to define the structure of the MS SQL table using the abovestated SQL Pass-Through command from PROC SQL:

```
  EXEC (ALTER TABLE &SQLtable ADD &SASvars) BY ODBC;                * define new structure;
```

Finally the redundant temporary Dummy column can be removed and the SQL Pass-Through facility is no longer needed:

```
  EXEC (ALTER TABLE &SQLtable DROP COLUMN &Dummy) BY ODBC;          * remove Dummy variable;
  DISCONNECT FROM ODBC;
```

FILL THE TABLE WITH DATA FROM THE SAS DATASET

Now that the structure of the MS SQL table has been built, the table has to be filled with content, data rows (observations). The only way it seems possible to feed just data (without a structure) from SAS to an MS SQL table is using the SQL INSERT command, which looks quite simple, but which needs a lot of preparation code as well:

```
  INSERT INTO &SQL_table SELECT * FROM &SAS_dataset;                * export data via ODBC;
```

The data is exported to the database table in the order of the SAS dataset. In any case it must be taken care of that the data is received, and thus sent, in the order of the columns defined in the MS SQL table. The safest way to force that is to sort the variables in the SAS dataset (as far as not yet sorted like that) in the order necessary, like:

```
%* Macro call: Create the variable list from the SAS dataset without attributes;
%LET Ordered = ; %* create the macro variable with empty contents;
%Var_List (DATA=&Dataset, StoreVar=Ordered, Delim=%STR( ) /* (no semicolon) */
%PUT Ordered=&Ordered; %* feedback of existing variables in SAS dataset;

* Force the variables in the dataset in that order by reordering them using RETAIN;
DATA &Dataset;
  RETAIN &Ordered;
  SET &Dataset;
RUN;
```

However, this may be redundant if the PROC CONTENTS output file has already been sorted by VARNUM as in the code presented on the previous page.

The data may now be fed into the MS SQL table in the correct order. The data corresponds to the right columns. However, the data itself doesn't know that, nor what type it actually is. Just like when defining the table structure and deducing information from DATE-, TIME- and DATETIME-variables to 'know' which columns are of which type, it is necessary to preprocess the data similarly. Note, that when letting SAS do the job, while CREATing, the user does not need to bother with it, except for taking care to associate the correct DATE-, TIME- and DATETIME-formats with the concerning variables. SAS then takes care of the numeric conversion.

In this case the numeric conversion has to be programmed explicitly, dependent on the variable's type. Therefore, before inserting the data into the MS SQL table it should go through a preprocessing data step. The data step takes care to multiply all values of DATE variables by 86400, the number of seconds per day, suited for a DATETIME variable, and to leave the values of TIME variables as they are (or rather limit them to values between 0 and 86400). The offset date for SAS and MS SQL is the same, 01jan1960, so no further adaptation is needed for that. DATETIME and DOLLAR variables are left unchanged, so the main adaptation occurs on the basis of a DATE format. The adaptation data step code is:

```
DATA &Dataset;
  SET &Dataset;
* = = = = = = = = = = = = = = = = Auto-Date Conversion = = = = = = = = = = = = = = = ;
*** Avoid the use of TIME variables (they will become DATETIME on date 01jan1960). ;
*** Avoid the use of DATE variables (they will become DATETIME with times 00:00:00);
*** DATE variables and related TIME variables should preferably be combined into   ;
*** a DATETIME variable as (86400 * Date + Time) or from the DHMS SAS function.    ;
*** Without a TIME variable the DATE should in any case be converted to DATETIME,  ;
*** by multiplying it by 86400 (seconds a day) as automatically done by SAS SQL    ;
*** CREATE or the data step. Below this is done explicitly using all date formats. ;
* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =;
  ARRAY ChronoVar _NUMERIC_;
  DO OVER ChronoVar; /* below no VFORMATN because of colon modifier */
    IF      (VFORMAT(ChronoVar) IN: (&DtTm_Fmt)) THEN /* exclude DateTime format */;
    ELSE IF (VFORMAT(ChronoVar) IN: (&Date_Fmt)) THEN ChronoVar = 86400 * ChronoVar;
```

```
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -;
* Likewise if it unfortunately occurs that any time variable has negative values   ;
* or values GE 86400 then the remaining after division by 86400 has to be taken,   ;
* if that is negative then 86400 should be added to obtain a value 0<=time<86400 . ;
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -;
    ELSE IF (VFORMAT(ChronoVar) IN: (&Time_Fmt)) THEN /* in all cases */
                         ChronoVar = MOD(ChronoVar,86400) + (ChronoVar LT 0)*86400 ;
  END;
* = = = = = = = = = = = = end of auto-date conversion = = = = = = = = = = = = =;
RUN;
```

It is even better if the user/programmer avoids the use of DATE- and TIME-variables in SAS and only applies DATETIME variables, enabling a one to one translation to MS SQL column values as the embedded comment indicates. Finally, as already said above, the data can be inserted into the MS SQL table:

```
  INSERT INTO &SQL_table SELECT * FROM &SAS_dataset;                  * export data via ODBC;
```

and PROC SQL may be finished:

```
QUIT;
```

CONCLUSIONS

Passing data from SAS to MS SQL tables at first sight seems easy once one has create/write access rights to the database. A simple PROC SQL CREATE or data step using ODBC suffices apart from the necessary login information. Once one's access rights are more limited, which is more likely or at least more recommended, it may become tricky to transfer data from SAS to the MS SQL database. This paper presents the case where one has only write access to one or more specifically identified, existing tables and yet wants to be able to use those tables for any data to transfer to the database. The strategy is to clear the tables from any information, data and structure and to rebuild the structure and the data from scratch from the SAS dataset. It proves that, even with very limited access rights, it is still possible to flexibly transfer any data and any structure to MS SQL tables.

The complete code of the strategy described is contained in a macro **%SAStSQLt** on the following pages. The macro (or its latest version) can also be found at the internet address below (ref. 4).

REFERENCES

1. SAS Institute Inc., 2004. SAS ® 9.1 SQL Procedure User's Guide. Cary, NC: SAS Institute Inc.
2. SAS/ACCESS® to External Databases: Wisdom for the Warehouse User. Judy Loren, Health Dialog Data Service, Inc., Portland, ME
3. Transact-SQL Syntax Conventions (Transact-SQL) © 2010 Microsoft Corporation. All rights reserved.
4. http://jim.groeneveld.eu.tf/software/SASmacro/SAStSQLt.zip or http://jim.groeneveld.eu.tf/SAStSQLt

CONTACT INFORMATION

Author Name :	Y. (Jim) Groeneveld	Work Phone :	+31 (0)73 523 6000
Company :	OCS Biometric Support	Fax :	+31 (0)73 523 6600
Address :	P.O. Box 3434	Email :	sasquestions@ocs-consulting.com
	5203 DK 's-Hertogenbosch	Web :	http://www.ocs-biometricsupport.com
	The Netherlands	Author's website :	http://jim.groeneveld.eu.tf


```
%*************************************************************************;
%* SAS macro SAStSQLt vs. 0.1.0 , 09 Aug 2010, by Jim Groeneveld, Netherlands *;
%*************************************************************************;
%*                                                                       *;
   %MACRO   SAStSQLt   /* transfer SAS dataset data to existing MS SQL table  */
/* ------------------------------------------------------------------------ */
/* Argument Default       Description and remarks      Required/Optional: R/O */
/* -------- --------      ------------------------------------------------- */
(  Dataset = /*_LAST_*//* Input SAS dataset name, including its libname     R */
,  MSSQLlib= MSSQLsrv  /* Name of library in SAS terms with ODBC engine     R */
,  SQLtable=           /* SQL table for which write permission to overwrite R */
,  DSN     =           /* Data Source Name of SQL database                  R */
,  UserName=           /* UserName for write access to SQL table            R */
,  Password=           /* Password for write access to SQL table            R */
,  Dummy   = _Dummy_   /* Dummy variable name to store in 'empty' SQL table R */
)/ DES     =             'copy SAS data to existing MS SQL table' /* STORE */  ;
%*                                                                       *;
```

```
%*************************************************************************;

  %LOCAL    /* Define internal macro variables explicitely local */
  Database  /* Name of SQL table in SAS terms: library.dataset   */
  DtTm_Fmt  /* Comprehensive list of all valid DATETIME formats  */
  Date_Fmt  /* Comprehensive list of all valid DATE     formats  */
  Time_Fmt  /* Comprehensive list of all valid TIME     formats  */
  MoneyFmt  /* List of all valid currency formats: only DOLLAR   */
  DataBase  /* Name of SQL table in SQL database in SAS terms    */
  SQLvars   /* List of all columns (variables) in SQLtable       */
  SASvars   /* List of all variables in SAS dataset              */
  ;

%*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~;
%* To interpret variables as representing special types like DATE, TIME and DATETIME ;
%* all concerning formats have to be checked in order to interpret variables correct.;
%*_____________________________________________________________________________;

%* Format type lists;
  %LET DtTm_Fmt = 'DATETIME' 'DATEAMPM';
  %LET Date_Fmt = 'DATE' 'DAY' 'DDMMYY' 'DOWNAM' 'EURDFD'
        'EURDFM' 'EURDFW' 'HDATE' 'HEBDAT' 'JULDAY' 'JULIAN' 'MINGUO' 'MMDDYY' 'MMYY'
        'MONNAM' 'MONTH' 'MONYY' 'NENGO' 'NLDATE' /*'NLDATM'*/ 'PDJULG' 'PDJULI' 'QTR'
         'WEEK' /*'WEEKDA'*/ 'WORDDA' 'YEAR' 'YYMM' 'YYMMDD' 'YYMON' 'YYQ';
  %LET Time_Fmt = 'TIME' 'TOD' 'HHMM' 'HOUR' 'MMSS' /*'TIMEAMPM'*/;
  %LET MoneyFmt = 'DOLLAR' ;
%* These lists may need adaptation with newer SAS versions;
%* Most macro variables are needed both inside and outside macro Var_List;

  LIBNAME &MSSQLlib ODBC DSN=&DSN USER=&username PWD=&password;  * MS SQL lib ODBC engine;
  %LET DataBase = &MSSQLlib..&SQLtable;

%* Create existing variable list from MS SQL database table (to be removed);
  %LET SQLvars = ; %* create the macro variable with empty contents;
  %Var_List (DATA=&Database, StoreVar=SQLvars, Delim=%STR(,)) /* (semicolon redundant) */
  %PUT SQLvars=&SQLvars; %* feedback of existing columns in MS SQL table;

%* Create variable list from SAS dataset;
  %LET SASvars = ; %* create the macro variable with empty contents;
  %Var_List (DATA=&Dataset, StoreVar=SASvars, Delim=%STR(,), Type=SQL) /* (no semicolon) */
  %PUT SASvars=&SASvars; %* feedback of existing variables (and MS SQL attr) in SAS dataset;

* Preprocess data, mainly DATE value conversion;
  DATA _Dataset;
    SET &Dataset;
* = = = = = = = = = = = = = = = Auto-Date Conversion = = = = = = = = = = = = = = ;
*** Avoid the use of TIME variables (they will become DATETIME on date 01jan1960). ;
*** Avoid the use of DATE variables (they will become DATETIME with times 00:00:00);
*** DATE variables and related TIME variables should preferably be combined into   ;
*** a DATETIME variable as (86400 * Date + Time) or from the DHMS SAS function.    ;
*** Without a TIME variable the DATE should in any case be converted to DATETIME,  ;
*** by multiplying it by 86400 (seconds a day) as automatically done by SAS SQL    ;
*** CREATE or the data step. Below this is done explicitly using all date formats. ;
* = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =;
    ARRAY ChronoVar _NUMERIC_;
    DO OVER ChronoVar; /* below no VFORMATN because of colon modifier */
      IF      (VFORMAT(ChronoVar) IN: (&DtTm_Fmt)) THEN /* exclude date format */ ;
      ELSE IF (VFORMAT(ChronoVar) IN: (&Date_Fmt)) THEN ChronoVar = 86400 * ChronoVar;
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -;
* Likewise if it unfortunately occurs that any time variable has negative values   ;
* or values GE 86400 then the remaining after division by 86400 has to be taken,   ;
* if that is negative then 86400 should be added to obtain a value 0<=time<86400 . ;
* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -;
      ELSE IF (VFORMAT(ChronoVar) IN: (&Time_Fmt)) THEN /* in all cases */
```

```
                          ChronoVar = MOD(ChronoVar,86400) + (ChronoVar LT 0)*86400 ;
    END;
* = = = = = = = = = = = = = end of auto-date conversion = = = = = = = = = = = = =;
  RUN;

  PROC SQL;
    CONNECT TO ODBC (DSN=&DSN USER=&username PWD=&password);    * Pass-Through Facility;
    EXEC (DELETE FROM &SQLtable) BY ODBC;                      * remove existing data;
    EXEC (ALTER TABLE &SQLtable ADD &Dummy FLOAT) BY ODBC;      * add a Dummy variable;
    EXEC (ALTER TABLE &SQLtable DROP COLUMN &SQLvars) BY ODBC;  * remove old structure;
    EXEC (ALTER TABLE &SQLtable ADD &SASVars) BY ODBC;          * define new structure;
    EXEC (ALTER TABLE &SQLtable DROP COLUMN &Dummy) BY ODBC;    * remove Dummy variable;
    DISCONNECT FROM ODBC;
    INSERT INTO &DataBase SELECT * FROM _Dataset;              * export data via ODBC;
  QUIT;

  PROC DATASETS NOLIST; DELETE _Dataset; RUN;

%MEND SAStSQLt;
%*_____________________________________________________________________________;

%* = = = = = = = = =;
%* Macro Var_List ; %* Determination of variable type for MS SQL table ;
%* = = = = = = = = =;
%MACRO Var_List (DATA=_LAST_, StoreVar=Var_List /* &Storevar should exist*/,
                Delim=%STR( ), Type=/*SAS/SQL*/);
  %LOCAL Contents;
  %LET Contents = ____List;
  PROC CONTENTS DATA=&Data OUT=&Contents (KEEP=NAME TYPE FORMAT VARNUM LENGTH) NOPRINT; RUN;
  PROC SORT DATA=&Contents; BY VarNum; RUN; * forcing variables in occurring order;
  DATA _NULL_;
    SET &Contents;
    LENGTH Attr $ 48;
    Attr = Name; * initialize with the variable (column) name;
    %IF (%UPCASE(&Type) EQ SAS) %THEN %DO;
      IF (Type EQ 2) THEN Attr = TRIM(Attr) || ' CHAR';
      ELSE IF (Type EQ 1) THEN Attr = TRIM(Attr) || ' NUM';
    %END;
    %ELSE %IF (%UPCASE(&Type) EQ SQL) %THEN %DO; %* deduce date and time variables;
      IF (Type EQ 2) THEN DO;
        IF (Length LE 8000) THEN Attr = TRIM(Attr) || ' VARCHAR(' ||
                                        TRIM(LEFT(PUT(Length,5.))) || ')';
        ELSE Attr = TRIM(Attr) || ' VARCHAR(MAX)';  * if allowed ; * at least 1024;
      END;
      ELSE IF (Type EQ 1) THEN DO;
        IF      (Format IN: (&DtTm_Fmt)) THEN Attr = TRIM(Attr) || ' DATETIME';
        ELSE IF (Format IN: (&Date_Fmt)) THEN Attr = TRIM(Attr) || ' DATETIME';
        ELSE IF (Format IN: (&Time_Fmt)) THEN Attr = TRIM(Attr) || ' DATETIME';
        ELSE IF (Format IN: (&MoneyFmt)) THEN Attr = TRIM(Attr) || ' MONEY';
        ELSE /* just numeric */              Attr = TRIM(Attr) || ' FLOAT';
      END;
    %END;
%*  else if &Type is neither SAS nor SQL then no attributes added to the names;
%* Build the column name+attr list in a single macro variable &StoreVar;
    CALL EXECUTE ('%LET &StoreVar = &&&StoreVar.&Delim.'||TRIM(Attr)||';');
  RUN;
%* remove initial delimiter &Delim;
  %LET &StoreVar = %SUBSTR(&&&StoreVar,2);

  PROC DATASETS NOLIST; DELETE &Contents; RUN;

%* = = = = = = ;
%MEND Var_List ;
%* = = = = = = ;
```