Using SAS With a SQL Server Database

M. Rita Thissen, Yan Chen Tang, Elizabeth Heath
RTI International, RTP, NC

ABSTRACT

Many operations now store data in relational databases. You may want to use SAS® to access and update information in SQL Server or other SQL-based systems. We have used SAS to manage and utilize project data in relational databases for a longitudinal survey. The survey applications use SAS through PROC SQL to read data from an existing SQL Server database or SAS macro language to update or insert data into SQL Server tables. The data-loading process also uses SAS formats and functions to clean data, which ensures compatibility with later SQL queries. We will provide examples of SAS code that perform these tasks.

INTRODUCTION

Longitudinal studies follow changes that occur over time in a population (or sample group).
In managing such studies, databases are needed for maintaining data collection events and subject contact data. We find it effective to use relational databases and SAS programs to track data collection activities, and we give here a brief discussion of the language elements and their usage under the Microsoft® Windows® operating system.

Structured Query Language (SQL) can be used to access and maintain data that are contained in relational databases. With Open Database Connectivity (ODBC) and PROC SQL, SAS users can create SAS datasets from, and manage the data contained in, relational databases. Before reviewing how SAS users can process SQL Server data, we need to briefly compare SQL and SAS syntax.

SQL LANGUAGE BASICS

Structured query language is a language used to define and manipulate database objects and the data they contain. As in other languages, each SQL statement contains a number of elements that must follow syntactic rules. Figures 1 and 2 show a simple SQL statement and its SAS language equivalent.

Relational tables are used to store data and are similar to SAS datasets. Tables are composed of rows (or records) that correspond to SAS rows or observations. Table rows are composed of columns (or fields), with each one named and defined as some data type. Columns are comparable to SAS variables. The fields named in the select statement are returned in the results of the query, much as the variables named in a KEEP= option are stored in the resulting dataset.

Figure 1. Simple SQL query

```
select Table1.CaseID, Table1.Name,
Table2.PhaseNum, Table2.Age, Table2.Health
from table1 inner join table2
on table1.CaseID = table2.CaseID
where CaseID ='101121'
order by Age
```

Figure 2. Equivalent SAS language code

```
proc sort data=dataset1 (keep = CaseID Name);
    by caseID;
run;

proc sort data=dataset2 (keep = CaseID PhaseNum
Age Health);
    by CaseID;
run;

data New;
    merge dataset1(in=in1) dataset2(in=in2);
    by CaseID;
    where CaseID ='101121';
     if in1 and in2;
run;

proc sort data=New;
    by Age;
run;
```

SQL statements allow you to specify a search condition. You can enter a clause containing a search condition to limit the rows of data you want the SQL Server to return. The SQL WHERE clause acts the same as the SAS language WHERE statement.

To combine multiple tables, an INNER JOIN is frequently stated with equal conditions for matching one or more columns of one table with those of another table. In SAS programs, this is achieved with the MERGE statement, IN= variables, an associated BY statement and conditions. Columns in relational databases do not need to have the same name to be used in the ON clause of the JOIN, although they commonly do share the same name.

With SQL, you can sort the returned results on any column value and specify an ascending or descending order through the ORDER BY clause, while in SAS programs, datasets are sorted with PROC SORT and BY statements.

SQL queries may employ multilevel names to uniquely identify a field: use database.owner.table.field in SQL to specify which database, owner and table the field belong to, if there is any ambiguity. In our example below, we use two-level naming, although one-level or multilevel names could be used if the fields were named uniquely. Multilevel names for SAS datasets are somewhat comparable, with LIBNAME at the top level followed by dataset name. However, the SAS language does not support the concept of owners, and variable names must be unique across any datasets that are merged, or else used as BY variables in the merge to yield a unique field in the resulting dataset.

USING PROC SQL TO READ SQL SERVER DATA

ODBC connections provide a programming interface to various types of databases. The ODBC connection permits data to be obtained from, modified in, or inserted into different types of ODBC databases from within SAS programs or other applications.

An ODBC connection must first be configured on the user's workstation for the relational database, through the appropriate Windows control panel. In configuring the connection, the server, database name, login, and password are established. Figure 3 shows how the connection is designated and activated through a SAS language program.

Figure 3. Sample PROC SQL statements

```
%let connectA = dsn="databaseName"                (A)
  uid="userID" pwd="password";
proc sql exec;                                    (B)
  connect to odbc (&connectA);                    (C)
  create table newSD2 as                          (D)
    select * from connection to odbc              (E)
      (select * from SQLtable where               (F)
      var1 = 'someCondition');

  create table secondSD2 as
    select * from connection to odbc
      (select * from diffSQLtable where
        var2 = 'someCondition');

disconnect from odbc;                             (G)
quit;                                             (H)
run;
```

A SAS macro variable (connectA) is shown in line (A) that provides the ODBC connection database name, user login, and password to be used in line (C). The login and password may be based on either Windows NT® or SQL Server authentication, but must match the ones used to establish the ODBC data source. For multi-user systems, each station must be configured with its own ODBC connection, and if Windows NT authentication is used, it is important to make sure that all users have access to the database.

SAS datasets can be created from SQL Server tables with the SQL procedure (PROC SQL). Please refer to SAS manuals about PROC SQL options that you may need. For example, if PROC SQL is submitted as part of a batch run, the EXEC option [line (B)] will allow the SQL procedure to continue to run subsequent SQL commands even if one SQL command fails.

As mentioned in the brief review of SQL language basics, a SQL table corresponds closely to a SAS dataset. In line (D), the CREATE TABLE statement will create a new SAS dataset named NewSD2. The new dataset, made via the ODBC connection in line (E), will be the result of the SQL statement in line (F).

For program correctness and syntactic checking, the SQL statement should be tested on the SQL Server prior to running PROC SQL. Testing the SQL statement may also reveal data value problems. For instance, in inserting or updating SQL Server table data, data values that contain an apostrophe (like O'No) may cause the SQL insert or update command to fail. Before running PROC SQL, such apostrophes could be removed or changed to a blank space in data values through techniques such as a SAS format assignment (with a formatted value of "'" = " ") or the SAS language compress function [revisedData = compress(orginalData, "'")].

The DISCONNECT statement closes the ODBC connection in line (G). In this example, two new SAS datasets are created from two different tables in the same SQL Server database before the ODBC connection is disconnected.

The QUIT statement in line (H) stops execution of the SQL procedure. The DISCONNECT and QUIT statements could be omitted if the SQL procedure were followed by another SAS procedure call or data step. If a DISCONNECT statement is not submitted, the ODBC connection is disconnected when the SQL procedure ends.

Even if you successfully run a SQL procedure, the SAS program may later fail if a variable has mismatched field lengths and types in datasets derived from SAS and SQL Server sources. To

avoid such problems, be sure to match variable attributes for data obtained from SAS and SQL Server sources.

EXPORT FROM SAS, IMPORT TO SQL SERVER

Data management operations may require the program to store new or updated data from SAS datasets to the SQL Server database. One approach writes the SAS data to a text file, which can then be imported into SQL Server tables by using one of several tools:

- Bulk Copy Procedure (BCP, bundled with SQL Server client tools)
- Data Transaction Services (bundled with SQL Server Enterprise Manager)
- Visual Basic® (VB) or other OLE-enabled language.

An example of BCP is given below, followed by a discussion of the other approaches.

Writing data from SAS to a text file is a common practice, using PUT statements. For example, see Figure 4, a text file with comma separators and fixed field lengths. The output from the SAS program is shown in Figure 5. We can use this output as input for BCP or one of the other import methods.

Figure 4. SAS program to write text file

```
%let comma=",";

data _null_;
set SASDemo;
file SASDemo;
put      CaseGrp  $CHAR6.    &comma
         CHFIRST  $CHAR019.  &comma
         CHINITL  $CHAR001.  &comma
         CHLAST   $CHAR020.  &comma
         CHAGE         004.  &comma
         CHDOB    $CHAR008.
;
run;
```

Figure 5. Text file output

```
101043,GEORGE              ,WASHINGTON          ,   2,11152001
101044,JOHN                ,ADAMS               ,   9,11122001
101036,THOMAS              ,JEFFERSON           ,   1,06292001
```

With BCP, we can add (insert) these rows into a table in a SQL Server database. The table must be defined in advance, and the input data must either match the field attributes exactly or be read in as text fields. In this example, the fields are all text. BCP requires a format file as a second input when reading data, such as the one in Figure 6.

The format file specifies the version of SQL Server compatibility (in this example, version 7.0), the number of table fields (6) and the characteristics of the text data to be imported. The first formatting line specifies that value #1 will be read as character data, with zero offset spaces, length six characters, separated by a comma, inputting the value to field #1 which has the field name "GroupID". In general, the value in the sixth formatting column will match the first. However, if your table has more fields than you need to use, you may mark them with a zero in the sixth format column, which tells BCP to skip that table field when inserting the data. Line 4, the field "Withdrawn," is skipped in this example. Note that you must indicate the absence of the separator comma if it will not be present.

Figure 6. BCP format file

```
7.0
6
1       SQLCHAR       0       6       ","          1     GroupID
2       SQLCHAR       0       19      ","          2     CHFirst
3       SQLCHAR       0       20      ","          3     CHLast
4       SQLCHAR       0       0       ""           0     Withdrawn
5       SQLCHAR       0       4       ","          5     ChAge
6       SQLCHAR       0       10      ","          6     ChDOB
```

Guidelines for preparation of BCP format files and instruction in the use of BCP can be found in Microsoft SQL Server Books Online and other sources.

Once the text file and BCP format file have been prepared, the insertion can take place on any station that runs SQL Server client software. The following BCP statement can be run from the command line or from within a command or scripting program.

```
bcp  SampleDB.dbo.SASData  in SASDemo.txt
  -fDemo.fmt   -SDemo4   -T
```

In this statement, the terms act as follows:

- **BCP** calls the bulk copy procedure (BCP)
- **SampleDB.dbo.SASData** specifies the SQL server database, table owner and table name
- **In** indicates that BCP is to accept input
- **SASDemo.txt** - the name of the input text file
- **-f** indicates that a format file should be found
- **Demo.fmt** is the name of the format file
- **-S** indicates that a server name will follow
- **Demo4** is the name of the server which holds the database
- **-T** indicates that BCP should use NT authentication.

Note that the flags (-f, -S and –T) are case sensitive. Other flags exist and can be used for other operations, as described in the literature.

Of course there are many other ways to import data from a text file into a table. Two common ones are through Microsoft's Data Transaction Service (DTS), which can be used interactively or as a scheduled task, or by creating a program in a language such as VB, which can take advantage of ODBC connections or Object Linking and Embedding (OLE) and the features of SQLOLEDB.dll.

When would you choose each tool? For batch-mode inserts, especially of large quantities of data, BCP is a good choice. However, BCP is not a tool for updating data; that is, if you wish to change values of fields in existing rows, you cannot use BCP. In that case, you would turn to DTS or a VB application. These tools require a deeper understanding of relational database management but offer greater power and flexibility.

WRITING FROM SAS TO SQL SERVER

Figure 7 provides sample SAS code that updates data in a SQL Server table called tbl_SASData. Lines marked in the figure perform the following tasks:

(A) initializes a macro variable to zero. This variable will be used as a counter, but would not be created in the data _null_ step if the input dataset had no observations

(B) selects variables to be output

(C) creates a macro variable Cmd# for each SAS observation. The macro variables form a macro array, in which the value is taken from the dataset variable SQLCmd. Each element of the macro array is a SQL statement which will be used to update a column named SASField for the appropriate table row. The row is selected according to a key field that matches the SAS variable VarVal with the table key value IDField.

(D) starts PROC SQL and then connects to the SQL Server database

(E) loops through the macro array, using the macro variable values as commands to the SQL Server database

(F) passes each command to the ODBC connection for execution

(G) disconnects the ODBC connection

(H) runs the macro program defined in lines (D) through (G).

Figure 7. SAS program to update SQL Server data

```
%let nobs = 0;                                          (A)

data _null_;
    set example
    (keep = CaseID VarVal) end=lastobs;                 (B)
    length SQLCmd $200;
    retain counter 0;
    counter = counter + 1;
    SQLCmd = "Update tbl_SASData set
    SASField = '" || compress(VarVal) || "'
    where IDField = '" || compress(CaseID) || "'
" ;
    if _N_ = 1 then do;
        * write one SQL command to the log for
        use in debugging;
        put SQLCmd=;
    end;
    call symput ("cmd" || compress(put(counter,
8.)), SQLCmd);                                          (C)
    * record the total number of observations in
a macro variable;
    if lastobs then do;
        call symput ("nobs", counter);
    end;
run;


%macro SQLDemo;
    proc sql exec;                                      (D)
    connect to odbc (&connect);

    %do i=1 %to &nobs;                                  (E)
        execute (
            &&cmd&i
            ) by odbc;                                  (F)
    %end;
    disconnect from odbc;                               (G)
    quit;
%mend;
%SQLDemo;                                               (H)
run;
```

SUMMARY

The sample code and commentary above may provide readers with a glimpse of the potential of PROC SQL. Heterogeneous systems such as those containing SAS programs and SQL Server databases are common in software systems for supporting survey research. Although they may seem difficult to manage at first, with the right tools an effective and practical system can be created.

ACKNOWLEDGEMENTS

We gratefully acknowledge support by R. Suresh and Jean Richardson.

CONTACT INFORMATION

M. Rita Thissen, rthissen@rti.org, (919) 541-6046
Yan Chen Tang , ytang@rti.org, (919) 541-7398
Elizabeth Heath, eah@rti.org (919) 485-2786

All authors can be reached at the mailing address:

RTI International
P.O. Box 12194
Research Triangle Park, NC 27709-2194.

TRADEMARK NOTICE