# or ## - how to reference SQL server temporary tables?
Xiaoqiang Wang, CHERP, Pittsburgh, PA

ABSTRACT
This paper introduces the ways of creating temporary tables in SQL Server, also uses some examples to illustrate the use of SQL Server temporary tables from a SAS® programmer's point of view.

Some techniques discussed also apply for other relational databases (we pick SQL Server because its syntax on temporary tables is more flexible) and can be initiated from other database clients.

INTRODUCTION
All major databases support temporary tables or temporary stored procedures. They're regarded as “temporary” because a temporary table/procedure will be dropped once the user session/connection (that creates it) is closed. I'd like to mention the temporary things because (1) it’s a neat solution sometimes because you don’t leave any trace in the database; (2) when your database administrator doesn't want you to mess up his database, those temporary things may be the last bullets out of your weapon.

Temporary tables/procedures are normally created via a special syntax but are referenced by their names just like normal tables/procedures. For example, we use “CREATE [[GLOBAL] TEMPORARY] TABLE …” to create a temporary table in ORACLE, and it’s referenced by its identifier/name once created. However, SQL Server is unique in the sense that it doesn’t require any keyword to be added into the CREATE TABLE/PROCEDURE statement. Instead, in SQL Server, a table/procedure name starting with a # sign indicates that it’s a temporary table/procedure. You'll realize the benefit of using a name convention instead of adding a keyword as a SAS user later.

The scope of a temporary table/procedure could be either global or session-specific. A global temporary/procedure can be referenced by other concurrent database sessions once created and before the session has created it closes, while a session-specific table/procedure is available only to the session that has created it. In SQL Server, a temporary table/procedure with name starting with ## is global, and it’s session-specific if its name starts with only 1 # sign. While in other databases, keyword GLOBAL is used in the creation of global table/procedures.

For now on, we’ll focus on SQL Server temporary tables.

CREATE A TEMPORARY TABLE
Because a SQL Server temporary table is distinguished only by one or two preceding # signs in table name, it can be created via either LIBNAME engines or the SQL Pass-Through facility.

To create a temporary table via a LIBNAME engine using a DATA step,

```sql
LIBNAME sqlsrv OLEDB INIT_STRING= "…" CONNECTION=SHARED;
DATA sqlsrv.'#ssn'n;
   ssn='123456789';
PROC PRINT;
RUN;
```

The dataset should print out if everything is fine. Please be aware that the INIT_STRING is omitted and the CONNECTION= option needs to be SHARED in the LIBNAME statement.

To create a temporary table via a LIBNAME engine in PROC SQL,

```sql
LIBNAME sqlsrv OLEDB INIT_STRING= "…" CONNECTION=SHARED;
PROC SQL;
   CREATE TABLE sqlsrv,'#ssn'n (ssn char(9));
   DESCRIBE TABLE sqlsrv,'#ssn'n;
QUIT;
```

You should see descriptions of the table in the log window.

To create a temporary table with the SQL Pass-Through facility,

```sql
PROC SQL;
   CONNECT TO OLEDB(INIT_STRING= "...");
   EXECUTE (
CREATE TABLE [#ssn] (ssn char(9));
BY OLEDB;
SELECT * FROM CONNECTION TO OLEDB(
    select * from [tempdb]..sysobjects
); DISCONNECT FROM OLEDB;
QUIT;

You should see some information about the table in the output window.

Now you see all syntaxes of creating a session-specific temporary table. To create a global temporary table, just replace the # sign with 2 # signs.

UPLOAD AN EXISTING TABLE AS A TEMPORARY TABLE

Often, we want to use an existing SAS dataset in SQL server. Now it becomes less convenient to do it in the SQL Pass-Through facility because multiple statements would be needed. But via a LIBNAME engine, we can do it with a DATA step, with PROC APPEND or PROCSQL.

Suppose we have a SAS dataset as follows.

DATA ssn;
do _n_ = 1 to 10000;
    ssn = put(floor(1e9*ranuni(1)), z9.);
    output;
end;

To upload a temporary table via a LIBNAME engine using a DATA step,

LIBNAME sqlsrv OLEDB INIT_STRING= "..." CONNECTION=SHARED;
DATA sqlsrv.'#ssn'n(INSERTBUFF=1000);
SET ssn;
RUN;
The INSERTBUFF needs to be set for better efficiency.

To upload a temporary table via PROC APPEND,

LIBNAME sqlsrv OLEDB INIT_STRING= "..." CONNECTION=SHARED;
PROC APPEND BASE=sqlsrv.'#ssn'n(INSERTBUFF=1000) DATA=ssn;
RUN;

To upload a temporary table with PROCSQL,

LIBNAME sqlsrv OLEDB INIT_STRING= "..." CONNECTION=SHARED;
PROC SQL;
    CREATE TABLE sqlsrv.'#ssn'n(INSERTBUFF=1000) AS
        SELECT * FROM ssn;
QUIT;

Those 3 methods are similar in performance.

We'll then see 3 examples where the use of temporary tables enhances efficiency.

EXAMPLE 1

Given patient administrative records in an SQL server table sf_2005 (scrsn, vizday, dob, sex, race, ethnic) with clustered index on scrsn. The table has 52,342,682 records. We just want a subset that the values of scrsn's belong to the SAS dataset ssn.

Without the help of a temporary table, we can submit a query like

LIBNAME sqlsrv OLEDB INIT_STRING= "...";
PROC SQL;
    CREATE TABLE xyz AS
        SELECT scrsn, vizday, dob, sex, race, ethnic
        FROM sqlsrv.sf_2005
        WHERE scrsn IN (SELECT ssn FROM ssn

Those 3 methods are similar in performance.

We'll then see 3 examples where the use of temporary tables enhances efficiency.

EXAMPLE 1

Given patient administrative records in an SQL server table sf_2005 (scrsn, vizday, dob, sex, race, ethnic) with clustered index on scrsn. The table has 52,342,682 records. We just want a subset that the values of scrsn's belong to the SAS dataset ssn.

Without the help of a temporary table, we can submit a query like

LIBNAME sqlsrv OLEDB INIT_STRING= "...";
PROC SQL;
    CREATE TABLE xyz AS
        SELECT scrsn, vizday, dob, sex, race, ethnic
        FROM sqlsrv.sf_2005
        WHERE scrsn IN (SELECT ssn FROM ssn

Those 3 methods are similar in performance.

We'll then see 3 examples where the use of temporary tables enhances efficiency.
If you turn on “options sastrace=’,d’ sastraceloc=saslog”, you’ll see all records in table sf_2005 were retrieved before they are filtered by the WHERE condition.

But with a little extra step of uploading SAS dataset ssn as an SQL Server temporary table, records can be filtered before getting out of SQL Server.

LIBNAME sqlsrv OLEDB INIT_STRING= "&connectstring" CONNECTION=SHARED;
PROC SQL;
    CREATE TABLE sqlsrv.'#ssn'n(INSERTBUFF=1000) AS
    SELECT * FROM ssn;
    CREATE TABLE xyz AS
    SELECT scrssn, vizday, dob, sex, race, ethnic
    FROM sqlsrv.sf_2005
    WHERE scrssn IN (SELECT ssn FROM sqlsrv.'#ssn'n);
QUIT;

We got enhanced performance as expected [1st method 3 minutes and a half, 2nd method less than 1 second].

EXAMPLE 2

Example 1 worked perfectly under SAS 9.1.3. But there seems to be a bug in SAS 9.2 to prevent our temporary table solution from outperforming the normal way. Here’s the work-around that would demand the help from a global temporary table.

LIBNAME sqlsrv OLEDB INIT_STRING= "&connectstring" CONNECTION=SHARED;
PROC SQL;
    CREATE TABLE sqlsrv.'##ssn'n(INSERTBUFF=1000) AS
    SELECT * FROM ssn;
    CONNECT TO OLEDB(INIT_STRING= "&connectstring");
    CREATE TABLE xyz AS
    SELECT * FROM CONNECTION TO OLEDB(SELECT scrssn, vizday, dob, sex, race, ethnic FROM sf_2005
    WHERE scrssn IN (SELECT ssn FROM [##ssn])
    );
    DISCONNECT FROM OLEDB;
QUIT;

The performance matches the 2nd method in Example 1 under SAS 9.1.3.

EXAMPLE 3

We can actually defer the reading of records outside of PROC SQL in Example 2.

LIBNAME sqlsrv OLEDB INIT_STRING= "&connectstring" CONNECTION=SHARED;
PROC SQL;
    CREATE TABLE sqlsrv.'##ssn'n(INSERTBUFF=1000) AS
    SELECT * FROM ssn;
    CONNECT TO OLEDB(INIT_STRING= "&connectstring");
    CREATE VIEW sf2005 AS
    SELECT * FROM CONNECTION TO OLEDB(SELECT scrssn, vizday, dob, sex, race, ethnic FROM sf_2005
    WHERE scrssn IN (SELECT ssn FROM [##ssn])
    );
    DISCONNECT FROM OLEDB;
QUIT;
DATA _null_
SET sf2005;
RUN;

The view in the example behaves like a Pass-Through query in Microsoft Access. It seems that the INIT_STRING in the CONNECT TO statement is retained with the view. When the view is requested to be opened, a new SQL Server session will be established to execute the Pass-Through query. On the other hand, the global temporary table [##ssn] is associated by the LIBNAME connection. It's accessible for the SQL Server session established by the view. If we issue a “LIBNAME sqlsrv CLEAR” command before the DATA _null_ step, the view will fail to be opened.

BEHIND THE SCENE

You may wonder why the view in Example 3 can be opened later after the DISCONNECT statement was executed in PROC SQL. You'll know the answer by simply describing view sf2005. The OLEDB connection string was stored together with the PROC SQL view, and that's why the view can be opened later.

```sql
397 PROC SQL;
398  DESCRIBE VIEW sf2005;
399  NOTE: SQL view WORK OF 2005 is defined as:
400  select * 
401 from connection to OLEDB
402 / * dBASE=OLEDB, connect options=INIT_STRING="Provider=SQLDEBB;1;Integrated Security=SSPI;Persist Security Info=True;Initial Catalog=Research;Data Source=chapthru" */
403  ( SELECT screen, vizeid, dob, sex, race, ethnic FROM sf_2005 WHERE screen IN ( SELECT ssn FROM [##ssn] ) );
```

Figure 1. Describe a PROC SQL view with pass-through query

This also behaves exactly like a pass-through query in Microsoft Access. Figure 2 is the SQL view of an Access pass-through query with a global SQL Server temporary table. The temporary table '##index_for_all' was created in VBA code. I don't know a way to use session-specific temporary table in an Access pass-through query.

Figure 2. SQL view of an Access pass-through query

Figure 3 below is the property sheet of the Access query. The connection string is stored with the query.

Figure 3. Property sheet of an Access pass-through query
CONCLUSION
Temporary tables are very helpful tools when data retrievers are prohibited to create regular tables in the database. It enhances the performance significantly. Because SQL Server temporary tables can be created in many ways, there’re many temporary table work-around to improve programming efficiency.

REFERENCES
SAS Institute Inc., SAS/ACCESS 9.1.3 for Relational Databases Reference,
SAS Institute Inc., SAS/ACCESS 9.2 for Relational Databases Reference,
SAS Institute Inc., SAS/ACCESS 9.1 Supplement for OLE DB,

ACKNOWLEDGMENTS
The author thanks the section chair for making this paper possible.

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

XIAOQIANG WANG
Center for Health Equity Research and Promotion
7180 Highland Dr (151C-H)
Pittsburgh, PA 15206
Work Phone: 412-954-5257
E-mail: xiw38@pitt.edu

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.
%let connectstring=%str(Provider=SQLOLEDB.1;Integrated Security=SSPI;Persist Security Info=True;Initial Catalog=Research;Data Source=vhapthrwh);
LIBNAME sqlsrv OLEDB INIT_STRING="&connectstring" CONNECTION=SHARED;
DATA sqlsrv.'#ssn'n;
   ssn='123456789';
PROC PRINT;
RUN;
LIBNAME sqlsrv CLEAR;
LIBNAME sqlsrv OLEDB INIT_STRING="&connectstring" CONNECTION=SHARED;
PROC SQL;
   CREATE TABLE sqlsrv.'#ssn'n (ssn char(9));
   DESCRIBE TABLE sqlsrv.'#ssn'n;
QUIT;
LIBNAME sqlsrv CLEAR;
PROC SQL;
   CONNECT TO OLEDB(INIT_STRING= "&connectstring");
   EXECUTE(
      CREATE TABLE [#ssn] (ssn char(9));
   ) BY OLEDB;
   SELECT name length=6, xtype, crdate, refdate FROM CONNECTION TO OLEDB(
      select left(name, charindex('_',name)-1) as name, xtype, crdate, refdate from [tempdb]..sysobjects
   );
   DISCONNECT FROM OLEDB;
QUIT;
DATA ssn;
   do _n_=1 to 10000;
      ssn = put(floor(1e9*ranuni(1)), z9.);
      output;
   end;
LIBNAME sqlsrv OLEDB INIT_STRING="&connectstring" CONNECTION=SHARED;
DATA sqlsrv.'#ssn'n(INSERTBUFF=1000);
   SET ssn;
RUN;
LIBNAME sqlsrv CLEAR;
LIBNAME sqlsrv OLEDB INIT_STRING="&connectstring" CONNECTION=SHARED;
PROC APPEND BASE=sqlsrv.'#ssn'n(INSERTBUFF=1000) DATA=ssn;
RUN;
LIBNAME sqlsrv CLEAR;
LIBNAME sqlsrv OLEDB INIT_STRING="&connectstring" CONNECTION=SHARED;
PROC SQL;
   CREATE TABLE sqlsrv.'#ssn'n(INSERTBUFF=1000) AS
      SELECT * FROM ssn;
QUIT;
LIBNAME sqlsrv CLEAR;

*example 1;
LIBNAME sqlsrv OLEDB INIT_STRING="&connectstring";
PROC SQL;
   CREATE TABLE xyz AS
      SELECT scrssn, vizday, dob, sex, race, ethnic
      FROM sqlsrv.sf_2005
      WHERE scrssn IN (SELECT ssn FROM ssn);
QUIT;

LIBNAME sqlsrv OLEDB INIT_STRING="&connectstring" CONNECTION=SHARED;
PROC SQL;
   CREATE TABLE sqlsrv.'#ssn'n(INSERTBUFF=1000) AS
      SELECT * FROM ssn;
   CREATE TABLE xyz AS
      SELECT scrssn, vizday, dob, sex, race, ethnic
      FROM sqlsrv.sf_2005
      WHERE scrssn IN (  
         SELECT ssn FROM sqlsrv.'#ssn'n
      );
QUIT;

*example 2;
*work-around the SAS 9.2 bug;
LIBNAME sqlsrv OLEDB INIT_STRING="&connectstring" CONNECTION=SHARED;
PROC SQL;
   CREATE TABLE sqlsrv.'##ssn'n(INSERTBUFF=1000) AS
      SELECT * FROM ssn;
   CONNECT TO OLEDB(INIT_STRING= "&connectstring");
   CREATE TABLE xyz AS
      SELECT * FROM CONNECTION TO OLEDB(
         SELECT scrssn, vizday, dob, sex, race, ethnic
         FROM sf_2005
         WHERE scrssn IN (  
            SELECT ssn FROM [##ssn]
         )
      );
QUIT;
DISCONNECT FROM OLEDB;

*example 3;
*deferred reading;
LIBNAME sqlsrv OLEDB INIT_STRING="&connectstring" CONNECTION=SHARED;
PROC SQL;
   CREATE TABLE sqlsrv.'##ssn'n(INSERTBUFF=1000) AS
      SELECT * FROM ssn;
   CONNECT TO OLEDB(INIT_STRING="&connectstring");
   CREATE VIEW sf2005 AS
      SELECT * FROM CONNECTION TO OLEDB(
         SELECT scrssn, vizday, dob, sex, race, ethnic
         FROM sf_2005
         WHERE scrssn IN (  
            SELECT ssn FROM [##ssn]
         )
      );
   CREATE VIEW sf2006 AS
      SELECT * FROM CONNECTION TO OLEDB(
         SELECT scrssn, vizday, dob, sex, race, ethnic
         FROM sf_2006
         WHERE scrssn IN (  
            SELECT ssn FROM [##ssn]
         )
      );
QUIT;
DISCONNECT FROM OLEDB;
DATA _null_
   SET sf2005 sf2006 OPEN=DEFER;
RUN;