Paper 2466-2018

Efficiently Join a SAS® Data Set with External Database Tables

Dadong Li, Michael Cantor, New York University Medical Center

ABSTRACT

Joining a SAS® data set with an external database is a relatively common procedure in research, and there are several methods to improve its performance. If only one external database table is involved, and the join is an inner join, using the dbkey data set option should be quick and easy. The second method is to generate a macro variable based on the SAS data set, which can then be used to abstract data. Since the macro variable works in both implicit and explicit SQL pass-through connections to the external database, this method is very flexible. If an analyst or an investigator has the permission, he could send the SAS data set into the database as a permanent table and use it like other database tables for data abstraction. However, in most cases, data analysts or investigators are not allowed to perform this kind of operation due to security or other reasons, limiting its use in the real work environment. The fourth method is to send a SAS data set into the external database as a temporary table, which in most cases does not need permission from database administrator and the temporary table is automatically dropped after the single PROC step. In a real-world scenario, the second and fourth methods, in most cases, are easily implemented, which makes them flexible to join a SAS data set with external database.

Keywords: SQL connections, data abstraction, data manipulation

INTRODUCTION

A common scenario during data abstraction for research projects is that the investigators want to join a small SAS data set with tables of an external large relational database, such as a SQL server or an Oracle database. Generally the performance of “straightforward” joining them together using data step or proc sql in SAS is very poor. The aim of this paper is to provide several solutions to this issue, and explain the pros and cons of each solution.

WHY THE PERFORMANCE OF DIRECTLY JOINING A SAS DATASET AND EXTERNAL DATABASE TABLES IS POOR

Suppose the investigators have a small list (work.patientlist) containing patient medical record numbers (MRNs) and other variables. They want to get more patients’ demographic information from a current large clinical database or data warehouse. It is easy to first think of the following “direct joining” procedure:

```
libname cogito odbc dsn='xxxxxxxxx' schema=dbo;  /*ODBC connection to
external database*/
proc sql;
create table final as
select  distinct
a.*,
b.sex,
b.ethnicity,
b.firstrace
from work.patientlist  a
     left join cogito.patientdim b
     on(a.mrn=b.primarymrn)
;
quit;
```

If the external database table is fairly small, the above syntax might work well. However if it is very large (most times it is), the above Proc SQL procedure will take a very long time to run since SAS is supposed to first abstract every row of the external database table , then join with the SAS data set. That is, in this case, millions of records of cogito.patientdim will be first extracted to a SAS session, then all but several hundred rows will be discarded since they do not match the MRN in the work.patientlist.

Furthermore, the above example is simple, that is, only one variable to match (mrn vs primarymrn), and only one external database table is involved. As the query becomes more complex, direct joining SAS data set and external database tables like above might take very long time to run.

There are several solutions to this issue.

SOLUTION 1. DBKEY DATA SET OPTION

The dbkey option will let SAS to build an internal WHERE clause to search for matches in the database management system (DBMS) table based on the key variable/column indicated in the dbkey option. Although the actual index on the variable/column in the database table is not required, based on the SAS 9.4 User's Manual, it is strongly recommended that an index exists for the key column in the underlying database table. Multiple key variables/columns can be used in the dbkey option.

Improper use of this option can decrease rather than increase performance.

If the query to join a SAS data set and external database table is simple, i.e. only one external database table is involved, the join is an inner join, and the join condition in the where clause is equality (such as a.mrn=b.priamrymrn), this should be a quick method to consider.

The following is an example.

```
proc sql;
create table demo_dbkey as
select distinct
a.*,
b.sex,
b.ethnicity,
b.firstrace
from work.patientlist a,
     cogito.patientdim(dbkey=primarymrn) b
where a.mrn_n=b.primarymrn
;
quit;
```

SOLUTION 2. MACRO VARIABLE METHOD

This method is actually composed of three steps: 1) generate a macro variable based on the SAS dataset; 2) extract the related information into SAS from the external database using the macro variable; 3) if necessary, join the original SAS dataset and the abstracted dataset within SAS. Here is an example.

```
proc sql noprint;
```

```
select distinct quote(strip(mrn),"'") into:list separated by ',',
from patientlist
;
quit;

proc sql;
create table demo as
select distinct
primarymrn,
sex,
ethnicity,
firstrace
from cogito.patientdim
where primarymrn in (&list)
;
quit;

proc sql;
create table final as
select  distinct
a.*,
b.sex,
b.ethnicity,
b.firstrace
from work.patientlist  a
     left join demo b
     on(a.mrn=b.primarymrn)
;
quit;
```

The macro variable can be easily generated and works in both implicit and explicit SQL pass-through connections to the external database. The SQL procedure in step 2 could be very complex (many tables and complex joining involved) and complex joining could also be implemented in Step 3, which makes this method very flexible and quick to be used. However, a macro variable has a size limit of 64k. If a very large macro variable needs to be generated from the SAS dataset, this method might not be convenient.

SOLUTION 3. WRITE THE SAS DATASET AS A PERMANENT TABLE INTO THE EXTERNAL DATABASE

If an analyst or an investigator is given the right by database administrator to load the SAS dataset into the external database, he could import the SAS dataset into the database (create a new table in the database), use it like other database tables, then most often drop it after your work is done. Here is an example:

```
proc sql;
create table cogito.temp_table03142017 as
SELECT  *,
monotonic() as row_number
 from patientlist
;

create table final as
select distinct
a.*,
b.sex,
b.ethnicity,
b.firstrace
from cogito.temp_table03142017  a
     left join cogito.patientdim b
    on(a.mrn=b.primarymrn)
;

drop table cogito.temp_table03142017       /*drop it after the job done*/
;
quit;
```

Note: If the SAS dataset is very large, it’s better to use the bulkload=yes option when importing it into the database.

Although this method is straightforward, in most cases, data analysts or investigators don’t have the right/permission to import a dataset into database due to security or other reasons, limiting its use in the real work environment.

SOLUTION 4. WRITE THE SAS DATASET AS A TEMPORARY TABLE INTO THE EXTERNAL DATABASE

This method allows for the importing of a SAS dataset as a temporary table, which in most cases does not need permission from database administrator and will automatically be dropped after the single proc step (though this might not be true for all DBMSs). Generally speaking, to join a temporary table (generated from SAS dataset) and a permanent table (original database table), one needs a libref for each table and these librefs must successfully share a global connection. To significantly improve performance, one must also set DBCOMMIT=0 like in the following example.

```
libname cogtemp odbc dsn='xxxxxxxxx' connection=global dbmstemp=yes
                     DBCOMMIT=0;
libname cogito odbc dsn='xxxxxxxxx' schema=dbo DBCOMMIT=0;

proc sql;
create table cogtemp.'#temp_patlist'n as
select distinct Pat_Enc_Csn_Id
from work.patientlist
;
create table patient_demo as
select distinct
pat.primarymrn,
pat.sex,
pat.firstrace,
pat.ethnicity,
enc.encounterepiccsn,
enc.datekey
from cogito.patientdim as pat
     inner join cogito.encounterfact as enc
          on (pat.patientkey=enc.patientkey)
     inner join cogtemp.'#temp_patlist'n as temp
          on (enc.encounterepiccsn=temp.Pat_Enc_Csn_Id)
```

```
;
quit;
```

CONCLUSION

Joining a SAS dataset with an external database is a relatively common procedure in research. There are several methods to improve the efficiency of this procedure, two of which (generating a macro variable and importing SAS dataset as a temporary table) in most cases are easily implemented and flexible.

REFERENCE

SAS/ACCESS Data Set Options : DBKEY=. Available at
http://www.okstate.edu/sas/v8/sashtml/accdb/z8-dbkey.htm

SAS 9.4 LIBNAME Engine for SAS Federation Server: User's Guide, Second Edition. Available at
http://support.sas.com/documentation/cdl/en/engfedsrv/70118/HTML/default/viewer.htm#p0ulrj2wyjwes2n1mgxd6u50l9xm.htm

SAS/ACCESS 9.4 for Relational Databases: Reference, Ninth Edition. Available at
http://support.sas.com/documentation/cdl/en/acreldb/69580/HTML/default/viewer.htm#p0he4t6yjfmkhpn16qrf0cdhllu6.htm

CONTACT INFORMATION

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

Dadong Li

Research IT DataCore

New York University Medical Center

hplcdadong@yahoo.com