

Joining A SAS® Dataset With A SQL Table Using Enterprise Guide

Gillis Guidry, Unisys Corporation, Baton Rouge, LA

Abstract

The following paper is offered as an alternative to the “normal” method of joining a SAS dataset with a SQL table using Enterprise Guide. By normal I mean the use of, for example, the task “Create Empty Query”. The software versions used were Enterprise Guide v1.3 and v2.0, and Microsoft SQL Server 2000. Whenever used, there was a marked improvement in data retrieval when using the explicit retrieval method rather than the implicit.

Problem

Users were complaining that Enterprise Guide was taking too long when joining a SAS dataset to a SQL table.

Cause

When joining a SAS dataset to a SQL database in Enterprise Guide (EG), SAS is using a libname engine, which is considered an “implicit pass-through”. In other words, SAS builds the specific SQL query and passes it to the SQL server on your behalf. EG is able to read the tables in the SQL database (named SQLDB for this paper) through the use of libraries. With this implicit pass-through query, SAS is reading all the data records from SQLDB specific to your joins and limitations. However it does not limit the data to the criteria (the where statement(s)) in your query. The implicit pass-through brings all the records into the SAS work area and lets SAS do the parsing. As you can see this will tie up the processor if the database has millions of rows of data. According to SAS this will be corrected in a later release.

When extracting SQL data in base SAS, the system uses what is called a more efficient “explicit pass-through” method. In other words, you explicitly code the SQL query and pass it to the SQL server. For base SAS, we use SAS/CONNECT with SAS/ACC-OLE DB to establish our access to the SQL server. Everything in the “inner select” (after the “select * from connection to oledb” in the examples below) is passed as is to this server. Using this method allows SQL to retrieve and return only what is needed. Executing code from EG’s code window can simulate running code in base SAS.

Solution

Since the SQL server does not recognize your SAS datasets, you must transfer them to a database table, and then use an explicit pass-through query to retrieve the desired results. A special SQL database TMPDB was developed to allow us to create, modify, and delete tables as needed for this purpose only. The examples below will run in EG’s code window and will mix the use of SAS implicit and explicit queries. One of our users created an implicit query that ran all night. After changes were made to accommodate the explicit method, the query ran in 17 minutes.

The solution is to move the SAS dataset to TMPDB and then utilize the base SAS method of retrieving SQL data. The following examples will show how to create a new table with indexes, and then build a PROC SQL explicit query utilizing this new table.

Step 1: Create a table in TMPDB

```
PROC DATASETS;  
copy in=SRCDATA out=TMPDB;  
select em_lipid_recips;  
RUN;
```

This will copy the SAS dataset Em_lipid_recips to the new SQL database TMPDB. SRCDATA is the libname where the SAS datasets are stored. TMPDB is both the SAS libname of the new SQL database and also the actual SQL database name. Use TMPDB when referencing the libname in an implicit query and TMPDB.dbo when referencing the database in an explicit query. The use of each is explained in the following paragraphs.

Step 2: Create an index on one or more of the fields

```
PROC SQL NOERRORSTOP;  
%include "!sasroot\Before_SQL.txt";  
EXECUTE(  
    create unique index idx1 on  
    TMPDB.dbo.em_lipid_recips (recip_id)  
    with ignore_dup_key  
    ) by oledb;  
%include "!sasroot\After_Sql.txt";  
QUIT;
```

The %include statements are used to provide the environment needed to connect to the SQL server and process your query. The "...Before_SQL.txt" (Appendix A) provides the "connect to oledb" init string and the "...After_SQL.txt" (Appendix B) provides the disconnect. Explicit queries must be inside these two statements. Only one set of %include's is needed per each PROC SQL statement.

The "EXECUTE (...) by oldeb" statement sends dynamic non-query SQL statements to the SQL server for processing. This one will create a unique index called idx1 on the field recip_id in the database table created above. Unique indexes guarantee that data in one column, or in a composite group of columns, remain unique for every row in a table. The IGNORE_DUP_KEY will not allow duplicates to be loaded into the table. However SQL Server does not allow the creation of a unique index on columns that already include duplicate values, whether or not the IGNORE_DUP_KEY is set. In other words, you cannot create a unique key on an existing SQL table with duplicate rows. If attempted, SQL Server displays an error message and the duplicates must be eliminated before a unique index can be created on the column(s).

The NOERRORSTOP will allow SQL to continue loading into a new table even though there are errors from the building of the unique index; example - " ERROR: Execute

error: Duplicate key was ignored". Without the NOERRORSTOP, SAS will stop processing with the first duplicate record encountered.

Step 3: Create an explicit SQL pass-through query

At this point it is very important to emphasize that if you use the EG query builder to create your query, EG will add "FORMAT ..." for each selected item and will add "SQLDB" to each SQL database table name. These two must be removed prior to using that query in the explicit mode. The following examples will show the differences between an implicit and an explicit query.

An example of an implicit SQL pass-through query (notice the libnames SQLDB and SRCDATA, the formatting properties, and no %include statements):

```
PROC SQL;
  SELECT RECIPINFO.Recip  FORMAT=11.,
         PAYMENTS.Date    FORMAT=8.,
         PAYMENTS.Payment FORMAT=8.2,
         MISC.Code         FORMAT=8.
  FROM SQLDB.RECIPINFO AS RECIPINFO
       INNER JOIN SQLDB.PAYMENTS AS PAYMENTS
         ON (RECIPINFO.Recip = PAYMENTS.Recip)
       INNER JOIN SQLDB.MISC AS MISC
         ON (PAYMENTS.Recip = MISC.Recip)
 WHERE (RECIPINFO.Recip IN (select em_lipid_recips.recip_id
                               from TMPDB.em_lipid_recips
                               as em_lipid_recips))
 AND (PAYMENTS.RecipDate BETWEEN 19990101 and 19990630)
 ;
QUIT;
```

An example of an explicit SQL pass-through query: (notice there are the %include statements and no SQLDB libnames to reference the database tables nor formatting properties.)

```
PROC SQL;

%include "!sasroot\Before_SQL.txt";

select * from connection to oledb
(
  SELECT RECIPINFO.Recip,
         PAYMENTS.Date,
         PAYMENTS.Payment,
         MISC.Code
  FROM RECIPINFO      AS RECIPINFO
       INNER JOIN PAYMENTS AS PAYMENTS ON (RECIPINFO.Recip = PAYMENTS.Recip)
       INNER JOIN MISC     AS MISC     ON (PAYMENTS.Recip = MISC.Recip)
 WHERE (RECIPINFO.Recip IN (select em_lipid_recips.recip_id
                               from TMPDB.dbo.em_lipid_recips
                               as em_lipid_recips))
 AND (PAYMENTS.RecipDate BETWEEN 19990101 and 19990630))
```

```
;
%include "!sasroot\After_Sql.txt";
QUIT;
```

The implicit query ran for 30 minutes. The explicit 5 seconds.

The remaining example is an explicit query. The “create table TMPDB.*tablename* as select * from connection to oledb “ is a PROC SQL pass-through statement that will create a SAS dataset named *tablename* from the results brought back from the connection to the SQL server. OLE DB is the Microsoft API used to access the data on the SQL server. The query that is passed to the SQL server is enclosed in parenthesis.

The first “create table” will create a SAS table Temp1003 in the database TMPDB with elig_id, race, and sex as the variables. Here we are using the SAS libname TMPDB to reference the database TMPDB.

The “execute” statement will create a unique index on this table for the variable elig_id.

The next “create table” will create a SAS table Final1003 in the SAS library SRCDATA with the variables recip, payment, and code, utilizing the Temp1003 table created above. As stated earlier, you must use TMPDB.dbo to reference the database inside the explicit SQL query or when creating indexes. The libname TMPDB is always used in an implicit query. A rule of thumb I follow is to use the TMPDB.dbo after the “select * from connection to oledb” and after the EXECUTE statement, and the libname TMPDB before the “select * from connection to oledb” and in the PROC DATASETS.

The PROC DATASETS below will copy the table Temp1003 from the SAS library TMPDB to the SAS library SRCDATA.

```
PROC SQL NOERRORSTOP;

%include "!sasroot\Before_SQL.txt";

create table TMPDB.Temp1003 as
select * from connection to oledb
(
    select elig_id, race, sex
    from eligible
    where race = '9' and sex='9'
);

execute(
    create unique index idx2 on TMPDB.dbo.Temp1003 (elig_id)
    with ignore_dup_key
    ) by oledb;

create table SRCDATA.Final1003 as
select * from connection to oledb
```

```

(
  SELECT Recip, Payment, Code
  FROM RECIPINFO AS RECIPINFO
  INNER JOIN PAYMENTS AS PAYMENTS
    ON (RECIPINFO.Recip = PAYMENTS.Recip)
  INNER JOIN MISC AS MISC
    ON (PAYMENTS.Recip = MISC.Recip)
  WHERE (RECIPINFO.Recip IN (select Temp1003.elig_id
                             from TMPDB.dbo.Temp1003
                             as Temp1003))
  AND (PAYMENTS.RecipDate BETWEEN 19990101 and 19990630)
)
;

%include "!sasroot\After_Sql.txt";

QUIT;

PROC DATASETS;
copy in=TMPDB out=SRCDATA;
select Temp1003;
RUN;

```

Conclusion

All the coding demonstrated in this paper has to be run in Enterprise Guide's code window. Whenever used, there was a marked improvement in data retrieval when using the explicit retrieval method rather than the implicit. Run times of course will greatly depend on the table sizes and server usage. As stated earlier, one user created an implicit query that ran all night. After changes were made to accommodate the explicit method, the query ran in 17 minutes - a significant difference.

References

SAS Institute, Inc. (1999), *SAS OnlineDoc Version 8*, Cary, NC:
SAS Institute, Inc.

SAS Technical Support, support@sas.com

SAS paper *Using The SAS/ACCESS Libname Technology To Get Improvements In Performance And Optimizations In SAS/SQL Queries*
<http://www2.sas.com/proceedings/sugi26/p110-26.pdf>

Contact Information

Your comments and questions are valued and encouraged.

Contact the author at:

Gillis.Guidry@unisys.com

Trademarks

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 registered trademarks or trademarks of their respective companies.

Appendix A

```
connect to oledb
(init_string="Provider=SQLOLEDB.1;Password=pswd;Persist
Security Info=True;User ID=userid;Initial Catalog=catalogname;Data
Source= SQLDB ");
```

Appendix B

```
disconnect from oledb;
```