PharmaSUG 2015 - Paper QT06

PROC SQL for SQL Die-hards

Jessica Bennett, Advance America, Spartanburg, SC
Barbara Ross, Flexshopper LLC, Boca Raton, FL

ABSTRACT

Inspired by Christianna William's paper on transitioning to PROC SQL from the DATA step, this paper aims to help SQL programmers transition to SAS by using PROC SQL. SAS adapted the Structured Query Language (SQL) by means of PROC SQL back with Version 6. PROC SQL syntax closely resembles SQL, however, there some SQL features that are not available in SAS. Throughout this paper, we will outline common SQL tasks and how they might differ in PROC SQL; as well as introduce useful SAS features that are not available in SQL. Topics covered are appropriate for novice SAS users.

INTRODUCTION

Transitioning from a SQL shop to a SAS shop, or vice versa, can be a challenging experience. Most SQL die-hards will find the PROC SQL procedure to be a life-saver. That is, until you realize that the minor syntax differences between the two can turn into major nuances, and that you can't actually copy and paste SQL code into SAS. In this paper, we will outline a few of the issues that we, as programmers, encountered when deemed with the task of transforming SQL logic into SAS code. This list is not all encompassing, but includes some of the key issues we have been facing while making the transition.

EXAMPLE 1. CREATING TEMPORARY TABLES

Both SAS and SQL allow users to create temporary tables in memory to reference later in their code. In SQL, a user must drop a temp table before they recreate it. If one tries to overwrite an existing temporary table, the code will error out stating that the table already exists. In SAS, the user does not have to drop the table before recreating. SAS forces the drop. SAS temporary tables are stored in the default WORK library. Everything in the WORK library is dropped when you end your SAS session. If you do not specify a library in SAS when creating a table, SAS automatically places that table into WORK.

The #TMP syntax tells SQL to remember that table in the current tab only. If one uses the ##TMP syntax, SQL will hold the temp table in memory across all tabs open in that session.

SQL Code:

```
IF OBJECT_ID('tempdb..#alldata') IS NOT NULL
DROP TABLE #alldata
SELECT *
INTO #alldata
FROM inputds_201411
```

SAS Code:

```
PROC SQL;
       CREATE TABLE alldata AS
       SELECT *
       FROM inputds_201411;
QUIT;
```

EXAMPLE 2. CREATING DYNAMIC/MACRO VARIABLES

To avoid having to retype the same parameter multiple times throughout a query, SQL and SAS both have options for creating dynamic variables. Most macro variables created in SAS are global and hold the value of that variable in memory as long as your session is open. SQL, however, only allows you to reference the variable while the query is processing. SAS does have local macro variables which are available only within that query. For more information on ways to create macro variables in SAS, see the "Ways of Creating Macro Variables" paper in the recommended reading section.

SQL Code:

```
DECLARE @FirstMonday date
```

```
SET @FirstMonday = '1-5-2014'
SELECT *
FROM inputds_201411
WHERE date > @FirstMonday
```

SAS Code:

```
%LET FirstMonday='05jan2014'd;
PROC SQL;
       CREATE TABLE tmp AS
       SELECT *
       FROM inputds_201401
       WHERE date>=&FirstMonday;
QUIT;
```

EXAMPLE 3. GLOBAL TABLES

SQL also offers the option to create global tables. Global tables are only held in memory while the query is running. Once the processing has completed, you are no longer able to call that table. Since it is so easy to create temporary tables in SAS, there is not a great need to create global tables. However if you would like to process data without creating output, SAS allows you to output to a _NULL_ destination. A _NULL_ table does not write any output which means it can't be referenced at all.

SQL Code:

```
DECLARE @TEST TABLE (
First_Name varchar(12) ,
Date_Created datetime
)
INSERT INTO @TEST VALUES
('Jessica' , '1-9-2015')
('Barbara' , '1-5-2015')

SELECT * FROM @TEST
```

SAS Code:

```
DATA _null_;
       SET inputds;
RUN;
```

EXAMPLE 4. CONVERT NUMERIC TO CHARACTER

To convert a variable from character to numeric (or vice versa), SQL has the CONVERT function. SAS does not support the CONVERT function. Rather you must use the PUT and INPUT functions. PUT applies a format to a value, which allows it to convert a numeric value to a character field. INPUT reads in a value based on an informat, which allows you to read in a character field as numeric. Note you must create a new variable if changing the character type. In PROC SQL, you can name the variable the same rendering it with a AS statement, but if converting in a SAS DATA step you must rename the original variable if you want the name to remain the same for the converted field.

SYNTAX:

```
PUT(source, format)
INPUT(source, informat)
CONVERT(data_type, expression)
```

In the example below, SSN is numeric and amount_owed is character. The z9 format adds leading zeros so that the outputted SSN value has a length of 9 digits (ex. 1234567 to 001234567). The dollar20 informat will read the numeric value of amount, ignoring the dollar sign (ex. $25.43 to 25.43).

SQL Code:

```
SELECT RIGHT('00000000' + CONVERT(VARCHAR(9), SSN), 9) as SSN ,
         CONVERT(numeric, REPLACE(amount, '$' , '') ) as amount
FROM inputds
```

SAS Code – No need to rename with PROC SQL:

```
PROC SQL;
       CREATE TABLE new AS
       SELECT        PUT(SSN,z9.) as SSN,
                     INPUT(amount,dollar20.) as amount
       FROM inputds;
QUIT;
```

SAS Code – Renaming with DATA Step:

```
DATA new;
       SET inputds (RENAME=(ssn=ssnn amount=amountc);
       SSN = PUT(SSNn, z9.);
       Amount = INPUT(amountc,dollar20.);
       DROP ssnn amountc;
RUN;
```

EXAMPLE 5. JOINING ON DIFFERENT VARIABLE TYPES

SQL automatically forces a merge with a character and numeric variable. With SAS, the variables being merged on must match. You can convert from character to numeric using the INPUT function and from numeric to character with the PUT function.

In the example below, inputds1 cust_id is character and inputds2 cust_id is numeric.

SQL Code:

```
SELECT *
FROM inputds1 a
INNER JOIN inputds2 b on b.cust_id = a.cust_id
```

SAS Code – Convert both to character:

```
PROC SQL;
       CREATE TABLE new AS
       SELECT *
       FROM inputds1, inputds2
       ON inputds1.cust_id = PUT(inputds2.cust_id,$5.);
QUIT;
```

SAS Code – Convert both to numeric:

```
PROC SQL;
       CREATE TABLE new AS
       SELECT *
       FROM inputds1, inputds2
       ON INPUT(inputds1.cust_id,best32.) = inputds2.cust_id;
QUIT;
```

EXAMPLE 6. SETTING TOGETHER SIMILAR DATASETS

Combining multiple datasets can be tedious when using SQL. SQL calls for you to select the variables desired from each dataset and join them with a UNION statement. Within SAS, PROC SQL functions the same way as SQL. In this case it is actually easiest to use a DATA step. With the DATA step, you can simply list the datasets to be merged with the SET statement. The SAS DATA step also has a shortcut for datasets with similar root names. Simply place a colon ":" after the root of the name and SAS will set together all the datasets that begin with that name.

SQL Code:

```
SELECT *
FROM inputds_201411
UNION
SELECT *
FROM inputds_201412
```

SAS Code – PROC SQL:

```
PROC SQL;
       CREATE TABLE alldata AS
       SELECT *
       FROM inputds_201411
```

```
UNION
       SELECT *
       FROM inputds_201412;
QUIT;
```

SAS Code – DATA Step:

```
DATA alldata;
       SET inputds_201411 inputds_201412;
RUN;

       -or-

DATA alldata;
       SET inputds:;
RUN;
```

EXAMPLE 7. USING DATE and DATETIME VALUES

Big difference between SQL and SAS is the way they store date and time values. In SQL all dates are stored with a date and time value. If you declare a datetime and do not add a time to the value, it will default to 0:0:0. With SAS, a variable can be a date or a datetime variable. Meaning dates can be stored without time values. When working with datetime values in SAS, you must use the DATEPART function in order compare it to a date value that has no time.

In the example below, date_created is a datetime variable.

SQL Code:

```
SELECT *
FROM inputd
WHERE CONVERT(DATE, Date_Created) >= '1-5-2014'

       -or-

SELECT *
FROM inputds
WHERE Date_Created >= '1-5-2014'
```

SAS Code:

```
PROC SQL;
CREATE TABLE tmp AS
       SELECT *
       FROM inputds
       WHERE DATEPART(date_created) >= '05jan2014'd;
QUIT;
```

EXAMPLE 8. INCREMENTING DATES

To increment a date in SQL, the DATEADD and DATEDIFF functions are commonly used. The DATEADD function takes a date value and increments it by a user specified interval. The DATEDIFF function is used to calculate the number of intervals between two dates. These functions are not available in SAS. SAS uses the INTCK and INTNX functions instead.

The INTNX function increments a date value by a given time interval and returns a date value. Useful for determining the start and end points of a period of time. The INTCK function returns the number of interval boundaries of a given kind between two date or datetime values.

Syntax:

```
INTNX(interval, start-from, increment,<alignment>);
INTCK(interval, start-date, end-date);
DATEADD(interval, increment int, expression);
DATEDIFF(interval, startingdate, ending_date);
```

SQL Code:

```
SELECT DATEADD(week, 2, LastPayDate) as NextPayDate
```

```
      , DATEDIFF(day, LastPayDate, getdate()) as DaysSincePaid
FROM inputds
```

SAS Code:

```
PROC SQL;
      CREATE TABLE tmp AS
      SELECT       INTNX('week',LastPayDate,2,'end') as NextPayDate,
                   INTCK('day',LastPayDate,today()) as DaysSincePaid
      FROM inputds;
QUIT;
```

EXAMPLE 9. RANK VARIABLES

The RANK function in SQL is often used to dedup a dataset by specifying the first observation to keep. RANK can also be used to number records within a group. There is no clear substitute for RANK in SAS. If you would like to dedup a dataset you can use the SORT Procedure. To number records within a group, the DATA step first. processing can be used.

SQL Code:

```
SELECT a.*
      , RANK() over (partition by cust_id order by date_created)     as n
      FROM inputds a
      Order by cust_id, date_created
```

SAS Code – Dedup dataset:

```
*First sort so the record you want to keep is listed first.
In this case the first record written;
PROC SORT DATA=inputds; BY cust_id date_created; RUN;
*The NODUPKEY keep will keep the first record for that BY group;
PROC SORT DATA=inputds OUT=outds NODUPKEY; BY cust_id; RUN;
```

SAS Code – Number within group;

```
*Sort records by group;
PROC SORT DATA=inputds; BY cust_id; RUN;
DATA outds;
      SET inputds;
      BY cust_id;
      IF first.cust_id THEN n=1;
      n+1;
RUN;
```

EXAMPLE 10. SELECTING N OBSERVATIONS

When exploring a new database, it is often helpful to be able to select a limited number of records from a dataset. To select n rows in SQL, one would select top n. Using SAS, you can use the OUTOBS= statement to limit the number of records shown. Just for comparison, in MySQL you must use the LIMIT statement.

SQL Code:

```
SELECT TOP 10 * FROM inputds_201411;
```

MySQL Code:

```
SELECT * FROM inputds_201411 LIMIT 10;
```

SAS Code:

```
PROC SQL outobs=10;
      CREATE TABLE tmp AS
      SELECT *
      FROM inputds_201411;
QUIT;
```

EXAMPLE 11. VARIABLE NAME LENGTH

It's worth noting that SAS is much more conservative than SQL when it comes to naming variables. If you are accessing a SQL database through a SAS libname, SAS will crop/rename your variable names to conform to their naming conventions.

Allowed variable name lengths:
- SAS - 32 characters
- Oracle - 30 characters
- SQL - 128 characters (116 characters in temporary tables)

Also SAS does not support variable names that start with a number. In SQL, you can't declare a name beginning with numbers, but you can rename them to. When accessing that table in SAS, SAS will prefix the name with an underscore (i.e. '2paydefault' renamed to '_2paydefault').

SQL allows for variable names with spaces as well. SAS Enterprise Guide (EG) also allows spaces, so these names will not be renamed when accessing your SQL tables through SAS EG. However, it is difficult to reference variables with spaces in their names. To have SAS rename variables names with spaces, you can use the system option VALIDVARNAME. Setting this to v7 will replace the spaces in the variable names with underscores ("_") making them easier to work with in SAS.

```
SAS OPTIONS VALIDVARNAME=v7;
```

EXAMPLE 12. SELECTING ROWS BASED ON A CREATED VARIABLE

SAS allows programmers the ability reference a calculated field in WHERE clause. This is unique to PROC SQL and is not available in SQL. When using SQL, one must type out the calculation if they chose to filter on a calculated variable.

SQL Code:
```
CREATE TABLE new AS
SELECT cust_id, date_created,
        (today()-date_created) as DaysBtw
FROM inputds
WHERE (today()-date_created) DaysBtw>1;
```

SAS Code:
```
PROC SQL;
       CREATE TABLE new AS
       SELECT cust_id, date_created,
               (today()-date_created) as DaysBtw
       FROM inputds
       WHERE CALCULATED DaysBtw>1;
QUIT;
```

EXAMPLE 13. CONCATENATING STRINGS

To concatenate strings, SQL uses the "+" operator while SAS uses the "||" operator. The SAS CAT function can be used instead of the double pipe operator as well. The CAT function in SAS is similar to the MySQL CONCAT function. SAS also has variations on the CAT function that will trim leading and trailing blanks, removing the need to use RIGHT/LEFT/TRIM functions. Popular SAS concatenation functions include: CATX (removes trailing and leading blanks and inserts delimiters), CATT (removes trailing blanks), and CATS (removes trailing and leading blanks).

SQL Code:
```
SELECT FirstName + ' ' + LastName As FullName FROM Customers
```

MySQL Code:
```
SELECT CONCAT(FirstName, ' ', LastName) As FullName FROM Customers
```

SAS Code - "||" Operator:
```
SELECT FirstName || ' ' || LastName As FullName FROM Customers
```

SAS Code - CATX Function:
```
SELECT CATX(' ',FirstName,LastName) As FullName FROM Customers
```

EXAMPLE 14. NOT EQUALS SHORTCUT

One small change between SAS and SQL is the shortcut notation for not equals. In SQL, you can type "!=". This notation is not recognized in SAS, you must use "^=". In both systems you can type out "NOT EQUAL" as well.

SQL Code:

```
SELECT *
FROM inputds
WHERE convert(date, date_created) != '1-5-2014'
```

SAS Code:

```
PROC SQL;
       CREATE TABLE new AS
       SELECT *
       FROM inputds
       WHERE date_created ^= '05jan2014'd;
QUIT;
```

CONCLUSION

This paper outlined many things we found different between SQL and PROC SQL while converting our SQL queries into SAS jobs. This paper is definitely not all encompassing, but is a jump start to understanding why some queries cannot be directly transferred over. As we continue to have our SQL and SAS programmers working closely together, we would like to expand on this topic and provide a more comprehensive paper that includes more differences between the two systems.

Think the major difference for a SQL programmer to understand while transitioning to SAS, is the way SAS creates tables. With SQL, many programmers have learned to create table structures then append onto those tables. This is not necessary in SAS. Also SAS does not require you to drop tables; it will force the drop and recreate the table for you. It is that approach to creating tables that will likely be the biggest thing for programmers transitioning between the languages to comprehend.

RECOMMENDED READING

- PROC SQL for DATA Step Die-hards, Christianna S. Williams, http://www2.sas.com/proceedings/forum2008/185-2008.pdf
- Ways of Creating Macro Variables, Kelley Weston, http://www.mwsug.org/proceedings/2011/tutorials/MWSUG-2011-TS04.pdf

CONTACT INFORMATION

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

Name: Barbara Ross
Enterprise: Flexshopper, LLC
City, State ZIP: Boca Raton, FL
E-mail: bmharlan@gmail.com
Web: www.bharlan.weebly.com

Name: Jessica Bennett
Enterprise: Advance America
City, State ZIP: Spartanburg, SC
E-mail: mary.jessica.bennett@gmail.com