A Closer Look at PROC SQL's FEEDBACK Option

Kenneth W. Borowiak, PPD, Inc., Morrisville, NC

ABSTRACT

The FEEDBACK option on the PROC SQL statement controls whether an expanded or transformed version of a query using terse notations is written to the SAS®log. This paper will review some of the documented features of this option and provide additional programming conventions that are explicitly stated when the option is enabled. It will be shown that the FEEDBACK option is an invaluable tool for understanding how PROC SQL processes a query and how it can be used as a code generator.

Keywords: PROC SQL, feedback, select *, natural joins, macro variables, operator mnemonics.

INTRODUCTION

The FEEDBACK option on the PROC SQL statement controls whether an expanded or transformed version of a query using concise notations is written to the SAS log. The simple query in Display 1 demonstrates some of the documented features of the FEEDBACK option, as per the SAS Online Doc dating back to at least Version 8.

Display 1 - Some of the Documented Features of the FEEDBACK Option

```
          option nosymbolgen ;
          %let cutoff=12;
          proc sql feedback ;
            create table class1 as
            select  *
            from    Sashelp.Class  /* No table alias specified */
            where   age < &cutoff.
                 or weight > 100
            ;
            quit ;

Partial SAS Log:

NOTE: Statement transforms to:

select CLASS.Name, CLASS.Sex, CLASS.Age, CLASS.Height, CLASS.Weight
  from SASHELP.CLASS
 where (CLASS.Age<12) or (CLASS.Weight>100);

NOTE: Table WORK.CLASS1 created, with 11 rows and 5 columns.
```

Though not shown, the SAS log contains the code sent to the compiler. In addition, the FEEDBACK option requests a rewritten version of the query to be written to the log, sans the CREATE TABLE statement of the query. The example demonstrates some of these documented transformations by the option:

- The SELECT * short-cut to select all variables from the source tables are explicitly stated.
- Macro variable references are resolved, which was CUTOFF in this example, despite the fact that the system option NOSYMBOLGEN was specified.
- Parentheses are shown around expressions to further indicate their order of evaluation, as shown in the WHERE clause.
- Comments are removed.

You should start to see the value of invoking the FEEDBACK option, as an explicitly written query with logical grouping of conditions can make debugging and reviewing queries easier. This can be especially helpful to SAS users who are less experienced with PROC SQL. As the default setting for PROC SQL is NOFEEDBACK and there is not a system option to set the FEEDBACK option, using a SAS Enhanced Editor abbreviation (2) to generate the PROC SQL statement with the FEEDBACK option and your other preferred settings is an efficient way to proceed [1].

VIEWS

The last of the documented features of the FEEDBACK option is that any PROC SQL view is expanded into the underlying query. Consider splitting of the SASHELP.CLASS data set into a table and view, as shown in Display 2.

Display 2 - Splitting SASHELP.CLASS into a Table and a View

```
proc sql ;
  /* Keep all fields except WEIGHT */
  create table part1 as
  select  *
  from    Sashelp.Class( drop=weight )
  ;

  /* Keep only NAME and WEIGHT fields */
  create view part2 as
  select  name, weight
  from    Sashelp.Class
  ;
  quit ;
```

Now reconstructing the table by joining to the two pieces using the primary key field NAME with the FEEDBACK option invoked is shown in Display 3.

[1] The options I have set for my PROC SQL statement include FEEDBACK and _method. See Lavery (3) for an excellent paper on the _method option.

Display 3 - Referencing an SQL View

```
          proc sql feedback ;
            create table class2 as
            select  *
            from    part1 as T1
                  , part2 as V2
            where   T1.name=V2.name
            ;
            quit ;

Partial SAS Log:
NOTE: Statement transforms to:

select T1.Name, T1.Sex, T1.Age, T1.Height, CLASS.Name, CLASS.Weight
  from WORK.PART1 T1,
       ( select CLASS.Name, CLASS.Weight
           from SASHELP.CLASS
        )
where T1.Name=CLASS.Name;

WARNING: Variable Name already exists on file WORK.CLASS2.
NOTE: Table WORK.CLASS2 created, with 19 rows and 5 columns.
```

You can see that the rewritten version of the query replaces the view referenced by the V2 alias with it's original definition. In addition, the V2 view alias is replaced with the source table (i.e. CLASS) when referencing fields.

If the FEEDBACK option is used in queries that references a view whose definition is based on another SQL view, the nested view is also expanded in the log. However, views created with a DATA step are not expanded. This is demonstrated in Display 4, where the view PART2 from the example in Display 3 is defined with a DATA step and the query is rerun.

Display 4 - DATA Step Views Are Not Expanded with FEEDBACK

```
          data part2 / view=part2 ;
            set Sashelp.Class( keep=name weight ) ;
            run ;

Partial SAS Log:
NOTE: Statement transforms to:

select T1.Name, T1.Sex, T1.Age, T1.Height, V2.Name, V2.Weight
  from WORK.PART1 T1, WORK.PART2 V2
 where T1.Name=V2.Name;

WARNING: Variable Name already exists on file WORK.CLASS2.
NOTE: View WORK.PART2.VIEW used (Total process time):
            real time           0.01 seconds
            cpu time            0.01 seconds

NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: Table WORK.CLASS2 created, with 19 rows and 5 columns.
```

CODE GENERATION - EXPANDING SELECT *

You may have noticed the queries in Display 3 and Display 4 resulted in a WARNING in the SAS log due to a collision in the variable namespace. This was caused by using the SELECT * variable list short-cut that was referencing the primary key NAME from both of the data sources. Some programming environments require 'clean' logs, where certain NOTEs, WARNINGs, and ERRORs are strictly prohibited. If namespace collisions in SQL queries are not permissible, then explicitly stating the relevant variables in the SELECT statement is a viable way to proceed. However, this may be cumbersome if there are numerous fields to list. The FEEDBACK option, though, could be used to generate the code for the query (except for the CREATE TABLE/VIEW statement) using terse conventions, such as SELECT *. The programmer could then copy an explicit version of the query from the log, remove the duplicate field reference(s), and then execute the query.

This two step process could be problematic if working with large amounts of data, as the query would need to be executed twice. However, you can request PROC SQL to generate a tranformed version of the query using the FEEDBACK without actually executing the query by using the VALIDATE statement. The VALIDATE statement in PROC SQL serves to check the syntax of the query without executing it. Display 5 shows the first step in the process, continuing with the example from Display 3.

Display 5 - Using the FEEDBACK Option and VALIDATE Statement for Code Generation

```
          proc sql feedback ;
             validate
             /* create table class2 as*/
             select  *
             from    part1 as T1
                   , part2 as V2
             where   T1.name=V2.name
             ;
             quit ;

Partial SAS Log:
NOTE: Statement transforms to:
validate
select T1.Name, T1.Sex, T1.Age, T1.Height, CLASS.Name, CLASS.Weight
  from WORK.PART1 T1,
       ( select CLASS.Name, CLASS.Weight
           from SASHELP.CLASS
        )
 where T1.Name=CLASS.Name;

NOTE: PROC SQL statement has valid syntax.
```

In order to use the VALIDATE statement properly the CREATE TABLE/VIEW statement should be excluded, hence the commented out line in the example. Now copy the expanded SELECT statement from the log into the editor and remove the redundant reference to NAME (i.e. T1.Name or CLASS.Name). For subsequent execution of the query, also remove or comment out the VALIDATE statement and add in the CREATE TABLE/VIEW statement. The log should no longer contain the WARNING about the duplicate field reference.

UNDOCUMENTED TRANSFORMATIONS

There are other transformations invoked by the FEEDBACK option that are not documented in the SAS Online doc. The query in Display 6 demonstrates some of these:

- The ? short-cut for the CONTAINS operator is expanded in the first in-line view V1.
- The & character for the AND logical operator is expanded in V1.
- The comparison operator GE is translated to >= in V1. Other comparison operator mnemonics, such LE, LT, GT, and EQ, are also translated.
- The |character for the OR logical operator is expanded in V2. The ! character would also be restated as OR.
- The join condition(s) involved with a natural join are explicitly written in the ON clause.
- The ORDER BY clause is augmented with the default ordering sequence, which is ascending. Despite the use of column positions with integers in the ORDER BY clause, the column names are not explicitly mentioned in the rewritten version of the query.

Display 6 - Undocumented Transformations of the FEEDBACK Option

```
                    proc sql feedback ;
                      create table undoc1 as
                      select   *
                      from     ( select  name "First Name", weight, age
                                 from    Sashelp.Class
                                 where   name ? 'a'
                                       & weight ge 10 ) as V1
                               natural left join
                               ( select  name "First Name", height, sex
                                 from    Sashelp.Class
                                 where   sex let 'G'
                                       | height lt 50 ) as V2
                      order by 1, 2
                      ;
                      quit ;

           Partial SAS Log:
           NOTE: Statement transforms to:

  select COALESCE(CLASS.Name, CLASS.Name) as name, CLASS.Height, CLASS.Sex,
CLASS.Weight, CLASS.Age
    from ( select CLASS.Name label='First Name', CLASS.Weight, CLASS.Age
             from SASHELP.CLASS
            where CLASS.Name contains 'a' and (CLASS.Weight>=10)
         ) as V1 left outer join
         ( select CLASS.Name label='First Name', CLASS.Height, CLASS.Sex
             from SASHELP.CLASS
            where (CLASS.Sex let 'G') or (CLASS.Height<50)
         ) as V2 on CLASS.Name=CLASS.Name
order by 1 asc, 2 asc;

           NOTE: Table WORK.UNDOC1 created, with 9 rows and 5 columns.
```

CODE GENERATION - NATURAL JOINS

Natural joins are another good candidate for using the FEEDBACK option for code generation. The SAS Online doc defines natural joins as a join that "selects rows from two tables that have equal values in columns that share the same name and the same type", where the join relation can be inner, outer, or if no common fields between the source tables, a Cartesian product. The use of natural joins in a production environment can be problematic, as the introduction of new fields into your queries can cause them to become part of the join conditions, which may not be intended. This problem can be avoided if you use explicit join conditions in your queries, which the FEEDBACK option can aid in generating. If you choose to use natural joins, then it is recommended you use the FEEDBACK option so that the join conditions can be examined in the log.

If you use the FEEDBACK option as a code generator for queries employing natural joins, then you should be aware of the attributes of the primary key fields in the resultant table. If you refer to Display 6 in the previous section, the field NAME was ostensibly recreated as COALESCE(V1.Name, V2.Name). Since a left join was used in the query, the values of NAME can only be contributed from the in-line view V1. The use of the COALESCE function is unnecessary, and more importantly, unfortunate, because it prevents the inheritance of the variable label from V1.NAME. If this is not the desired outcome, then you will want to modify the generated code for the primary key fields in the SELECT statement to meet your specifications.

CONCLUSION

The FEEDBACK option provides an explicit version of an SQL query that uses short-cuts and terse notations in the SAS log. This provides valuable insight as to how SQL processes a query, as well as an aid for less experienced SQL users who are required to maintain or modify the code. The FEEDBACK option can also be used as a code generator so that an explicit version of the query is the body of the program, rather than in the log. This paper reviewed some of the documented transformations of query by the FEEDBACK option and exposed some new ones. Hopefully, curious readers who make this option a standard part of their PROC SQL queries will discover new transformations.

REFERENCES

[1] Borowiak, Kenneth W., "Variable List Short-Cuts in PROC SQL", NESUG 2006
`http://www.nesug.org/proceedings/nesug06/cc/cc05.pdf`

[2] Grant, Paul, "Creating Code Templates in the SAS Enhanced Editor using Abbreviations and User Defined Keywords"
`http://support.sas.com/resources/papers/proceedings09/077-2009.pdf`

[3] Lavery, Russ, "The SQL Optimizer Project: _Method and _Tree in SAS9.1", SUGI 30
`http://www2.sas.com/proceedings/sugi30/101-30.pdf`

ACKNOWLEDGEMENTS

The author would like to thank Jenni Borowiak, Bill Deese, Jim Worley, Kipp Spanbauer, Mike Kwee and Ted Cherico for their insightful comments in reviewing this paper. Ron Fehd assisted with the formatting of this paper using LaTeX.

DISCLAIMER

The contents of this paper are the work of the author and do not necessarily represent the opinions, recommendations, or practices of PPD, Inc.

CONTACT INFORMATION

Your comments and questions are valued and encouraged.

Name: Ken Borowiak
Enterpise: PPD, Inc.
Address: 3900 Paramount Parkway
City, State, Zip: Morrisville, NC 27560

Email: Ken.Borowiak@ppdi.com
Ken.Borowiak@gmail.com
Phone: (919) 456-5373