PharmaSUG 2013 – Poster # P015

SQL SUBQUERIES: Usage in Clinical Programming

Pavan Vemuri, PPD, Morrisville, NC

ABSTRACT

A feature of PROC SQL which provides flexibility to SAS users is that of a **SUBQUERY**. A SUBQUERY is a query inside a query. A SUBQUERY can be further classified into an INLINE VIEW, correlated SUBQUERY or noncorrelated SUBQUERY depending on the usage inside the query. This paper explores the types of SUBQUERIES, their merits and demerits in comparison to a DATA step approach with examples relevant to clinical trials programming.

INTRODUCTION

A SUBQUERY can also be called an INNER QUERY as it is essentially a query inside a query. For the remainder of this paper unless specified as correlated or noncorrelated, the word SUBQUERY refers to an inner query. A SUBQUERY can be used in the WHERE, HAVING, SELECT or FROM clauses, although they are most commonly used in the WHERE and FROM clauses. They cannot be referenced outside the SQL step they are being used, and the resultant column/columns that a SUBQUERY selects cannot be saved as a table but merely used with the outer query. The occurrence of a SUBQUERY inside the outer query determines its classification into an INLINE VIEW, correlated SUBQUERY and noncorrelated SUBQUERY. In the pharmaceutical industry and in the clinical trial world there are many instances where SUBQUERIES can be used. Some of these scenarios that may be encountered on a day to day basis are illustrated below. To evaluate efficiency relative to a traditional DATA step approach, real time taken by the programs to run and the number of DATA/SQL steps required to achieve the result are compared. Test datasets LABS and ECG with 25000 to 30000 observations respectively have been used as input datasets. The numbers of variables are restricted to the ones that are in use for the test case. SAS v9.2 on Intel i5 processor with Windows operating system is used.

INLINE VIEW

As discussed, there are mainly two types of SUBQUERY. One of it is an INLINE VIEW that occurs inside a FROM clause. A programmer can use all aspects of a general SQL query except for the ORDER BY clause. Below are two examples that demonstrate the efficiency and flexibility of an INLINE VIEW.

Deriving the Baseline flag in a Lab dataset

Consider deriving the baseline flag in a lab dataset. In this scenario the aim is to flag the last occurring lab test before the treatment start date as baseline record per subject, visit, category and test. The approach is twofold where the baseline value and flag are derived in the INLINE VIEW that is used in the join operation with outer query. The input dataset structure (not the test dataset used for comparison) is shown below which is followed by program code and resultant dataset.

SUBJID	TRTN	TRTSTDT	AVISITN	PARAMCAT	PARAMCD	LBDT	AVAL
1	1	20MAR2011	100	Chemistry	TEST1	10MAR2011	10
1	1	20MAR2011	100	Chemistry	TEST1	21MAR2011	15
1	1	20MAR2011	100	Chemistry	TEST1	05FEB2011	1
1	1	20MAR2011	100	Chemistry	TEST1	01JAN2011	21
1	1	20MAR2011	100	Chemistry	TEST2	15MAR2011	1
1	1	20MAR2011	100	Chemistry	TEST2	05MAR2011	14
1	1	20MAR2011	100	Chemistry	TEST2	06FEB2011	25
2	2	02FEB2011	100	Chemistry	TEST2	10JAN2011	12
2	2	02FEB2011	100	Chemistry	TEST2	15FEB2011	16
2	2	02FEB2011	100	Chemistry	TEST2	30JAN2011	19
2	2	02FEB2011	150	Chemistry	TEST2	01JAN2011	23
2	2	02FEB2011	150	Chemistry	TEST2	15JAN2011	21

Table 1. Input Dataset LABS

Program using Inline view

```
Proc sql;
Create table LAB_Flag as select labs.*, ilv.basefl from labs as lb
left join

(select max(LBDT) as basedt format date9.,aval as base, trtn, avisitn,
paramcat, paramcd, subjid, "Y" as BASEFL     from labs(where=(lbdt<trtsdt))
group by subjid, trtn, avisitn, paramcat, paramcd having max(lbdt) = lbdt) as ilv

on lb.subjid=ilv.subjid and lb.trtn=ilv.trtn and lb.avisitn =ilv.avisitn and
lb.paramcat = ilv.paramcat and lb.paramcd= ilv.paramcd and lb.lbdt = ilv.basedt and
lb.aval = ilv.base  ;

quit;
```

SUBJID	TRTN	TRTSTDT	AVISITN	PARAMCAT	PARAMCD	LBDT	AVAL	BASEFL
1	1	20MAR2011	100	Chemistry	TEST1	10MAR2011	10	Y
1	1	20MAR2011	100	Chemistry	TEST1	21MAR2011	15	
1	1	20MAR2011	100	Chemistry	TEST1	05FEB2011	1	
1	1	20MAR2011	100	Chemistry	TEST1	01JAN2011	21	
1	1	20MAR2011	100	Chemistry	TEST2	15MAR2011	1	Y
1	1	20MAR2011	100	Chemistry	TEST2	05MAR2011	14	
1	1	20MAR2011	100	Chemistry	TEST2	06FEB2011	25	
2	2	02FEB2011	100	Chemistry	TEST2	10JAN2011	12	
2	2	02FEB2011	100	Chemistry	TEST2	15FEB2011	16	
2	2	02FEB2011	100	Chemistry	TEST2	30JAN2011	19	Y
2	2	02FEB2011	150	Chemistry	TEST2	01JAN2011	23	
2	2	02FEB2011	150	Chemistry	TEST2	15JAN2011	21	Y

Table 2. Final Dataset LAB_FLAG

As discussed, the baseline flag, the baseline lab date and the baseline value are selected inside the INLINE VIEW (which is shown in bold). The function MAX and GROUP BY clause in the INLINE VIEW help in choosing the last lab date before the treatment start date and keeping only one record per treatment, subject, visit, category and test. The resultant table aliased “ilv” is then used in the left join to get the desired output. When the program is run using the test LABS dataset, which has 30,000 observations, run time observed is **0.21 seconds**. The alternative

approach, using a Data step, is explored below to achieve the same result. The run time observed in this case is **0.24 seconds**.

Program using DATA step approach

```
proc sort data = labs out = labs1;
by trtn subjid avisitn paramcat paramcd lbdt;
where lbdt <trtsdt;
run;

data temp_labs;
set labs1;
by trtn subjid avisitn paramcat paramcd lbdt;
if last.paramcd;
run;

proc sort data =labs;
by trtn subjid avisitn paramcat paramcd lbdt;
run;

data Labs_flag;
merge labs(in=a) temp_labs(in=b);
by trtn subjid avisitn paramcat paramcd lbdt;
if a and b then basefl ='Y';
run;
```

Although the run time remained about the same, the key aspect to note is that using an INLINE VIEW the result could be achieved in one SQL step instead of multiple SORT procedures and DATA steps. The capability of an INLINE VIEW to use various functions that can be used in the outer query makes the code concise compared to a traditional approach.

Adding calculated observations to ECG dataset

Here is another scenario where an INLINE VIEW can be used. Consider deriving of an ECG dataset from SDTM data. For this example assume that the same test is repeated multiple times per visit. The aim here is to add one more row which will be the average of all the test values for that test per subject, treatment and visit. The approach is similar to the example discussed above. The input dataset, program and the resultant dataset are as follows.

SUBJID	TRTN	AVISITN	PARAMCD	EGSTRESN
1	1	100	TEST1	10
1	1	100	TEST1	15
1	1	100	TEST1	1
1	1	100	TEST1	21
1	1	100	TEST2	1
1	1	100	TEST2	14
1	1	100	TEST2	25
2	2	100	TEST2	12
2	2	100	TEST2	16
2	2	100	TEST2	19
2	2	150	TEST2	23
2	2	150	TEST2	21

Table 3. Input Dataset EG

Program using INLINE VIEW

```
Proc sql;
create table ECG as select eg.* from eg
       outer  union corr
(select "Average" as Dtype , round(avg(egstresn),.01) as
aval,subjid,trtn,paramcd,avisitn  from Eg  group by subjid, trtn, avisitn, paramcd)
order by subjid, avisitn, paramcd;
quit;
```

SUBJID	TRTN	AVISITN	PARAMCD	EGSTRESN	DTYPE	AVAL
1	1	100	TEST1		Average	11.75
1	1	100	TEST1	10		
1	1	100	TEST1	15		
1	1	100	TEST1	1		
1	1	100	TEST1	21		
1	1	100	TEST2		Average	13.33
1	1	100	TEST2	1		
1	1	100	TEST2	14		
1	1	100	TEST2	25		
2	2	100	TEST2		Average	15.67
2	2	100	TEST2	12		
2	2	100	TEST2	16		
2	2	100	TEST2	19		
2	2	150	TEST2	23		
2	2	150	TEST2	21		
2	2	150	TEST2		Average	22

Table 4. Final Dataset ECG

In the program the Average is calculated in the INLINE VIEW (shown in bold) and only the average value per subject, per test is kept using the GROUP BY clause. This is then set with the parent data using the OUTER UNION operator. Using the Test EG dataset of about 25000 observations run time observed is **0.37 seconds**. Alternatively the same result can be obtained using DATA step as shown below. The run time observed in this case is **0.54 seconds**.

Program using Data step approach

```
proc sort data = eg out = eg_srt;
by subjid trtn avisitn paramcd;
run;

proc means data = eg_srt mean noprint;
var egstresn;
output out = egmean mean = aval;
by subjid trtn avisitn paramcd;
run;

data ecg;
length DTYPE  $7;
set eg(in=a) egmean(in=b);
if b then Dtype ='Average';
if aval ne . then aval = round(aval,0.01);
run;
```

```
proc sort data= ecg;
by subjid avisitn paramcd;
run;
```

Similar to the previous Labs example, although the run times are not different, it is evident that the number of steps required is noticeably less (only one step) in an INLINE VIEW compared to the traditional approach.

CORRELATED and NONCORRELATED SUBQUERY

An inner query that is occurring in the SELECT, WHERE or HAVING clause is called a SUBQUERY (correlated/noncorrelated). It is most commonly seen in the WHERE clause. Another noteworthy difference between a SUBQUERY (correlated/noncorrelated) and an INLINE VIEW is that a SUBQUERY (correlated/noncorrelated) can select only one column. A SUBQUERY is called correlated when the inner query cannot be executed independently but rather is dependent on the outer query. A SUBQUERY is called noncorrelated when both the outer query and inner query are executed independently.

Selecting the desired observation

Consider a daily situation where subjects in Dataset A but not in Dataset B are to be selected. This can be achieved by both a correlated and a noncorrelated SUBQUERY. The dataset structure (not the test datasets used in comparison) and respective code are as shown below.

Subjid
001
002
003
004
005
006

Table 5. Input Dataset A

Subjid
006
007
008
009
005
006

Table 6. Input Dataset B

Program Using a correlated SUBQUERY

```
Proc sql;
Create Table OnlyA as select * from A where not exists
          (select * from B where A.subjid =B. subjid);
Quit;
```

The keyword "not exists" before the SUBQUERY (shown in bold) in the WHERE clause creates a correlation with the outer query, hence called correlated SUBQUERY. Each row evaluated in the outer query is passed to the inner query as the contributing table A is only called in the outer query but referenced in the inner query. Since the inner query is run for each row evaluated by the outer query it requires **increased processing time**. The real time observed is **23 seconds.** Hence, the correlated SUBQUERY is not the practical choice.

Program using a noncorrelated SUBQUERY

```
Proc sql;
Create Table OnlyA as select * from A where a.subjid not in
(select distinct subjid from B);
Quit;
```

Alternatively

```
Proc sql;
Delete from table A where subjid in (select distinct subjid from B);
Quit;
```

In the above noncorrelated SUBQUERY both the outer query and the inner query (shown in bold) is executed independently. Hence the word noncorrelated SUBQUERY. Using test datasets of ECG and LABS the run time observed is **0.14 seconds.** The resultant dataset of both the correlated and the noncorrelated SUBQUERY is below.

Subjid
001
002
003
004

Table 7. Dataset OnlyA

The above resultant dataset can also be achieved by data step programming and is shown below. The run time observed is **0.16 seconds.** Similar to the program using an INLINE VIEW, a noncorrelated SUBQUERY yields concise and efficient code.

Program using traditional Data step approach

```
proc sort data = A out = A_UNQ(keep = subjid) nodupkey;
by subjid;

proc sort data = B out = B_UNQ(keep = subjid) nodupkey;
by subjid;
run;
```

```
data onlyA;
merge A_UNQ(in=a) B_UNQ(in=b);
by subjid;
if a and ^b;
run;
```

INLINE VIEW vs. CORRELATED and NONCORRELATED SUBQUERY

An inner query occurring in the FROM clause is called an INLINE VIEW. It can be used to select multiple columns and can have all the functionality of a regular SQL query except for the ORDER BY statement. An inner query that is occurring in the SELECT, WHERE or HAVING clause is called a SUBQUERY (correlated or noncorrelated) and can only select a single column. The selection of the inner query is determined by the task at hand. If one single column is of interest then a noncorrelated SUBQUERY is ideal. If the task involves multiple columns an INLINE VIEW is ideal.

CONCLUSION

SQL SUBQUERY can be classified into INLINE VIEW, CORRELATED SUBQUERY and NONCORRELATED SUBQUERY depending on their usage inside the outer query. They can be utilized to develop concise and efficient programs. The above shown examples explain and showcase some of the common scenarios in clinical trial programming where SUBQUERY can be used. It is also seen from the above examples that using INLINE VIEW and noncorrelated SUBQUERY improved efficiency by reducing the length of the code noticeably, while a correlated SUBQUERY was not ideal due to longer runtime. To conclude, it is worthwhile to explore the SQL SUBQUERIES and to incorporate them in daily programming tasks to improve programming efficiency.

ACKNOWLEGEMNETS

Many thanks to my wife for her help in editing and PPD colleagues, esp. Ken Borowiak for their support and guidance.

CONTACT INFORMATION

For comments and suggestions

Pavan Vemuri

Sr.Programmer/Analyst

PPD, Morrisville NC

Pavan.Vemuri@ppdi.com

919-456-4534