WORKING WITH SUBQUERY IN THE SQL PROCEDURE
Lei Zhang, Domain Solutions Corp., Cambridge, MA
Danbo Yi, Abt Associates, Cambridge, MA

ABSTRACT
Proc SQL is a major contribution to the SAS®/BASE system. One of powerful features in SQL procedure is subquery, which provides great flexibility in manipulating and querying data in multiple tables simultaneously. However, subquery is the subtlest part of the SQL procedure. Users have to understand the correct way to use subqueries in a variety of situations. This paper will explore how subqueries work properly with different components of SQL such as WHERE, HAVING, FROM, SELECT, SET clause. It also demonstrates when and how to convert subqueries into JOINs solutions to improve the query performance.

INTRODUCTION
One of powerful features in the SAS SQL procedure is its subquery, which allow a SELECT statement to be nested inside the clauses of another SELECT, or inside an INSERT, UPDATE or DELETE statement, even or inside another subquery. Depending on the clause that contains it, a subquery can select a single value or multiple values from related tables. If more than one subquery is used in a query-expression, the innermost query is evaluated first, then the next innermost query, and so on. PROC SQL allows a subquery (contained in parentheses) to be used at any point in a query expression, but user must understand when and where to use uncorrelated/correlated subqueries, non-scalar and scalar subqueries, or the combinations. This paper will discuss the subquery usage with following sections:

- Subquery in Where clause
- Subquery in Having clause
- Subquery in From clause
- Subquery in Select clause
- Subquery in SET clause
- Subquery and JOINs

The examples in this paper refer to an imaginary set of clinical trial data, but the techniques used will apply to other industries as well. There are three SAS data sets in the example database: Data set DEMO contains variables PATID (Patient ID), AGE, and SEX. Data set VITAL contains variables PATID, VISID (Visit ID), VISDATE (Visit date), DIABP and SYSBP (patient’s diastolic/ systolic blood pressure which is measured twice at a visit). Data set VITMEAN contains patient’s mean diastolic/ systolic blood pressure at each visit recorded by doctors in variables PATID, VISID, VISDATE, MDIABP and MSYSBP. The source SAS codes to create the three data sets are listed in the appendix.

SUBQUERY IN THE WHERE CLAUSE
We begin with WHERE clause because it is the most common place to use subqueries. Consider following two queries:

Example 1: Find patients who have records in table DEMO but not in table VITAL.
Proc SQL;
Select patid from demog
Where patid not in (select patid from vital); 
Result:

<table>
<thead>
<tr>
<th>PATID</th>
</tr>
</thead>
<tbody>
<tr>
<td>150</td>
</tr>
</tbody>
</table>

Example 2: Find patients who didn’t have their blood pressures measured twice at a visit.
Proc SQL;
select patid from vital T1
where 2>(select count(*)
from vital T2
where T1.patid=T2.patid
and
T1.visid=T2.visid); 
Result:

<table>
<thead>
<tr>
<th>PATID</th>
</tr>
</thead>
<tbody>
<tr>
<td>110</td>
</tr>
<tr>
<td>140</td>
</tr>
</tbody>
</table>

In the example 1, The SELECT expression in the parentheses is an un-correlated subquery. It is an inner query that is evaluated before the outer (main) query and its results are not dependent on the
values in the main query. It is not a scalar subquery because it will return multiple values.

In the example 2, the SELECT expression in the parenthesis is a correlated subquery because the subquery refers to values from variables T1.PATID and T1.VISID in a table T1 of the outer query. The correlated subquery is evaluated for each row in the outer query. With correlated subqueries, PROC SQL executes the subquery and the outer query together. The subquery is also a scalar subquery because aggregate function COUNT(*) always returns one value with one column for each row values from the outer query at a time.

Multiple subqueries can be used in the SQL WHERE statement.

**Example 3:** Find patients who have the maximum number of visits.

```
Proc SQL;
   Select distinct patid 
   from vitmean as a 
   Where not exists 
      (select visid from vitmean 
       Except 
       Select visid from vitmean as b 
       Where a.patid=b.patid);
```

Result:

<table>
<thead>
<tr>
<th>PATID</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
</tr>
</tbody>
</table>

Below is a nested correlated subquery solution to example 3.

```
Proc SQL;
   Select distinct patid from vitmean as a 
   where not exists 
      (select distinct visid from vitmean 
       Except 
       Select visid from vitmean as b 
       Where a.patid=b.patid) 
       and 
      (select visid from vitmean as c 
       Where a.patid=c.patid 
       and 
       b.visid=c.visid));
```

This solution is using a double negative structure. The original question can be put in another way: Find each patient for whom no visit exists in which the patient concerned has never had. The two nested subqueries produce a list of visits for whom a given patient didn’t have. The main query presents those patients for whom the result table of the subquery is empty. SQL procedure determines for each patient separately whether the subquery yields no result.

In the WHERE clause, IN, ANY, ALL, EXISTS predicates are allowed to use any type of subqueries (see example 1 and 3 above); However, only scalar subqueries can be used in relation operators (>, =, <, etc), BETWEEN, and LIKE predicates. One way to turn a subquery to into a scalar subquery is using aggregate functions. Consider Example 4 below, which two scalar subqueries are used in the BETWEEN predicate.

**Example 4:** Find patients whose age is in the average age +/- 5 years.

```
Proc SQL;
   Select patid 
   from demog 
   where age between 
      (select Avg(age) 
       from demog)-5 
      and 
      (select avg(age) 
       from demog)+5;
```

Result:

<table>
<thead>
<tr>
<th>PATID</th>
</tr>
</thead>
<tbody>
<tr>
<td>110</td>
</tr>
</tbody>
</table>

SUBQUERY IN THE HAVING CLAUSE

Users can use one or more subqueries in the Having clause as long as they do not produce multiple values. It means that the most outer subquery in the Having clause must be scalar subquery, either uncorrelated or correlated. The following query is an example,

**Example 4:** Find patients who didn’t have maximum number of visits.

```
Proc SQL;
   Select patid 
   From vitmean as a 
   Group by patid 
      (select count(*) 
       from vitmean 
       Except 
       Select count(*) 
       from vitmean as b 
       Where a.patid=b.patid) 
       and 
      (select count(*) 
       from vitmean as c 
       Where a.patid=c.patid 
       and 
       b.visid=c.visid); 
```

Result:

<table>
<thead>
<tr>
<th>PATID</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PATID</th>
</tr>
</thead>
<tbody>
<tr>
<td>110</td>
</tr>
</tbody>
</table>
Having count(visid) <(
   select
   max(cnt)
   from
   (select
    count(visid)
    as cnt
    from vitmean
    group by patid)
);
(select patid, visid from VITAL) AS T2
on T1.visid=T2.visid
and T1.patid < T2.patid
Group BY T1.PATID, T2.PATID
Having count(*)=(Select count(*)
From vital as T3
Where
T3.PATID=T1.PATID)
and
count(*)= (Select Count(*)
From Vital as T4
Where
T4.PATID=T2.PATID);

RESULT:

<table>
<thead>
<tr>
<th>PATID</th>
<th>PATID1</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>130</td>
</tr>
</tbody>
</table>

SUBQUERY IN THE SELECT CLAUSE

SAS SQL can use a scalar subquery or even a correlated scalar subquery in the SELECT clause. If the subquery results are an empty, the result is a missing value. This feature lets you write a LEFT JOIN as a query within SELECT clause. Here is an example.

Example 7: Compute the difference between recorded mean of SYSBP from VITMEAN table and calculated mean of SYSBP from VITAL for each patient and visit.

Proc SQL;
Select a.patid, a.visid,
a.msysbp -
(select avg(sysbp)
from vital as b
where a.patid=b.patid and
a.visid=b.visid)
as diff
from vitmean as a;

Result:

<table>
<thead>
<tr>
<th>PATID</th>
<th>VISID</th>
<th>DIFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>110</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>110</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>120</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>120</td>
<td>2</td>
<td>34</td>
</tr>
<tr>
<td>120</td>
<td>3</td>
<td>7.5</td>
</tr>
<tr>
<td>120</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>130</td>
<td>1</td>
<td>-0.5</td>
</tr>
<tr>
<td>130</td>
<td>2</td>
<td>-0.5</td>
</tr>
<tr>
<td>130</td>
<td>3</td>
<td>-0.5</td>
</tr>
<tr>
<td>140</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>140</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

The subquery in the SQL clause is a correlated scalar subquery. It will calculate average systolic blood pressure based on VITAL table for each patient visit from table VITMEAN. The main query then use the result to calculate the difference between recorded systolic blood pressure and calculated systolic blood pressure for each patient visit. The process will be repeated until all patients in table VITMEAN are checked.

SUBQUERY IN THE SET CLAUSE

You can use scalar subquery and/or correlated scalar subquery in the SET clause of the Update statement to modify a table based on itself or another table. Here is an example.

Example 8: Modify the mdiabp and msysbp in the VITMEAN using values calculated from table Vital;

proc sql;
UPDATE VITMEAN as a
Set mdiabp=(select avg(b.diabp)
from VITAL as b
where a.patid=b.patid and
a.visid=b.visid),
msysbp=(select avg(c.sysbp)
from VITAL as c
where a.patid=c.patid and
a.visid=c.visid);
SUBQUERY AND JOINS

Some users find subquery difficult to read, and consequently fear that subqueries are difficult to use. Indeed, many subqueries can be replaced by one or more join queries. However, if you can’t construct an equivalent join query, the only alternative will be to use a series of queries and pass the results from one query to another. This is obviously less efficient than using subqueries. Subqueries can perform wonders of selection and comparison so that one set of statements and nested subqueries can replace several non-subquery statements. This improve efficiency and could even effect performance.

Generally whether using a join or a subquery is simply a matter of individual preference: they can often be used interchangeably to solve a given problem. Users will find that subqueries exceed the capabilities of join for certain queries. In some cases, a subquery is the shortest route to the results a user need. However, if users can find an equivalent join query for a subquery, it can replace the subquery to improve the performance. For example, the JOIN solution to the example 1 is:

```
proc SQL;
  select a.patid
  from demog as a
    left join
      (select distinct patid
       from vital) as b
    on a.patid=b.patid
  where b.patid is NULL;
```

and the JOIN solution to Example 7 is.

```
Proc SQL;
  select a.patid, b.visid,
       max(msysbp)-avg(b.sysbp) as diff
  from vitmean as a
    left join
      vital as b
    on a.patid=b.patid
      and a.visid=b.visid
  group by a.patid, b.visid;
```

In some situation, it may be worthwhile to look into rewriting such subqueries as JOINSs, but the development and maintenance of such codes may have them less desirable than subqueries because JOINS solution may improve the performance at the cost of losing readability.

CONCLUSIONS

We summarize the usages of subquery in the different SQL components as follows:

<table>
<thead>
<tr>
<th>SQL Components</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relation Operator (&lt;, =, &gt;, etc)</td>
<td>Scalar uncorrelated/correlated subquery</td>
</tr>
<tr>
<td>Between, Like</td>
<td>Scalar uncorrelated/correlated subquery</td>
</tr>
<tr>
<td>IN, ANY, ALL, SOME, Exists</td>
<td>Uncorrelated/Correlated subquery</td>
</tr>
<tr>
<td>FROM Clause</td>
<td>Uncorrelated subquery</td>
</tr>
<tr>
<td>Having Clause</td>
<td>Scalar uncorrelated/correlated subquery</td>
</tr>
<tr>
<td>Select Clause</td>
<td>Scalar uncorrelated/correlated subquery</td>
</tr>
<tr>
<td>SET clause</td>
<td>Scalar uncorrelated/correlated subquery</td>
</tr>
</tbody>
</table>

SAS and SAS/BASE mentioned in this paper are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

REFERENCES


AUTHOR’S ADDRESS

Lei Zhang
DomianPharma Inc.
10 Maguire Road
Lexington, MA 02140, USA
Tel: 781-778-3880
Fax: 781-778-3700
e-mail: lzhang@domainpharma.com

Danbo Yi
Abt Associates Inc.
55 Wheeler Street
Cambridge, MA 02138-1168, USA
Tel: 617-349-2346
Fax: 617-520-2940
e-mail: danbo_yi@abtassoc.com
APPENDIX: EXAMPLE SAS DATA SETS:

Data demog;
  input patid $1-3 age 5-6 sex $ 8;
  datalines;
  100 34 M
  110 45 F
  120 67 M
  130 55 M
  140 40 F
  150 38 M
;   run;

data vital;
  input patid $ visid $ visdate mmdyy10. diabp
  sysbp;
  format visdate mmdyy10.;
  datalines;
  100 1 01/12/97 66 110
  100 2 01/19/97 67 110
  110 3 01/23/97 68 120
  110 1 02/03/97 68 110
  110 3 02/17/97 66 115
  120 1 04/05/97 65 105
  120 2 04/12/97 64 107
  120 3 04/19/97 65 105
  120 4 05/23/97 91 107
  130 1 02/12/97 66 110
  130 2 03/19/97 66 109
  130 3 04/26/97 68 119
  140 1 02/03/97 68 110
  140 2 02/12/97 66 115
;   run;

Data vitmean;
  input patid $ visid $ visdate mmdyy10. mdiabp
  msysbp;
  format visdate mmdyy10.;
  cards;
  100 1 01/12/97 67 110
  100 2 01/19/97 66 110
  100 3 01/26/97 68 120
  110 1 02/03/97 68 110
  110 3 02/17/97 66 115
  120 1 04/05/97 65 105
  120 2 04/12/97 64 107
  120 3 04/19/97 65 105
  120 4 05/23/97 91 107
  130 1 02/12/97 66 110
  130 2 03/19/97 66 109
  130 3 04/26/97 68 119
  140 1 02/03/97 68 110
  140 2 02/12/97 66 115
;   run;