THE PARKE-DAVIS ASSESSMENT PROGRAM
AN EXAMPLE USING SQL IN SAS® PROGRAMMING

Russell Newhouse, Warner-Lambert/Parke-Davis

Introduction

In antiinfective clinical studies, various pieces of data are needed to determine the general condition of a patient. After these pieces are collected, the outcome of the patient is assessed based on these data. Therefore, it is necessary to define assessment scores based on changes in the variables measured so that a patient can be classified as either a responder or nonresponder to treatment.

Three different assessments are done. The Clinical Assessment is an overall assessment based on the patient’s signs and symptoms of disease (fever, cough, rash, etc.). A second assessment is based on the presence or absence at the follow-up visit of each pathogen that was cultured at baseline, the Pathogen Assessment. The Bacteriological Assessment is an overall assessment of each patient based on the presence or absence at follow-up of all of the pathogens cultured from that patient at baseline. (The Pathogen and Bacteriological Assessments are based on the same data, but are different interpretations of the data.)

The process of assigning these assessments had been performed by using listing tables of the data and manually assigning an assessment based on the outcome definitions. However, a large number of studies were planned to occur within the same time frame, and a more efficient method of assigning assessments was desired. So it was decided to computerize the entire assessment process.

This paper will present some of the more interesting SQL queries from the Parke-Davis Assessment Program. These range from examples of a simple query that uses a mathematical expression to create a new variable, to using nested queries and subqueries, to using the coalesce function to interleave data, to appending data sets using an outer union corresponding.

Creating a Data Set With a New Variable

The first step in assessing a patient is the selection of one observation per defined visit: baseline, short term follow-up (STFU) and long term follow-up (LTFU). This selection is based on when the patient is scheduled to be examined according to the study protocol. For example, a protocol could define the visits as follows: the baseline visit is the observation closest to, but not after, the day a patient starts drug (study day 1); the STFU is 5-9 days posttherapy; and the LTFU is 4-6 weeks posttherapy. When a range of days for a visit is given, the observation closest to the midpoint of the range is selected. If a follow-up visit was outside of the specified range, the data were still useful for that visit; however, the visit would be excluded from any evaluable patient analysis, but would be included in an intent-to-treat analysis (ITT). For this reason, the visit definitions for the ITT are widened from the evaluable analysis definitions. Figure 1 presents a graphical representation of the visit definitions using the example time frames given above.

Figure 1
Each observation is categorized into one of the three visits. This is performed by calculating the number of days posttherapy for each observation and then categorizing the observation based on the study day for the baseline visit or the number of days posttherapy for the follow-up visits, using the ITT visit definitions (or "windows").

In an early version of the assessment program, this entire process was performed by a long SQL query. The query first defined a new data table using a subset of data from two data sets, calculated the number of days posttherapy (postdose), added a new variable (visit), then updated the table to contain the values for visit. While the resulting query worked, it was extremely slow. It was faster to have a query create the table containing only the needed variables to determine visit (including calculating postdose), and then use a data step to add visit to the table.

**Example 1**

```
proc sql;
create table obsdate as
    select distinct a.prot, a.trial, a.ptno, sdobs, sdend, (sdobs-sdend) as postdose
    from sd.micro as a, sd.dosing as b
    where a.prot=b.prot & a.trial=b.trial & a.ptno=b.ptno;
quit;
```

The query takes patient identification information and the study day of each observation (sdobs) from the permanent data set sd.micro, which contains the culture information for each patient. It also takes the last study day that the patient received drug (sdend) from the permanent data set sd.dosing, which contains various dosing information. A new variable is then calculated using sdobs and sdend which is given the alias postdose. This is the number of days posttherapy for the observation.

**Filtering Data Using a Nested Query**

Before the visit selection can take place, it might be necessary to filter out unwanted observations. This filtering may be the result of taking a concurrent antibacterial drug and the presence of data collected both during therapy and posttherapy.

If a patient is taking a nonprotocol antibacterial drug during the study, it is desired to use data collected before the start of the new drug. These data are desired since they are considered to be "clean", that is, they haven't been affected by the new drug. However, if there are no data for a visit before the start of the new drug, then the data after the start of the drug will not be filtered out of the selection process (The observation would be used in the ITT analysis, but not in the evaluable analysis).

In the STFU window, it is possible that a patient will have observations collected both during therapy and posttherapy. In fact, it is possible that an observation collected during therapy may be closer to the selection target day than any posttherapy observations. However, it is the posttherapy observations that are of interest. Therefore, if there are any posttherapy observations within the STFU window for a patient (after taking into account new antibacterial drugs), any observations that were collected during therapy are filtered out of the selection process.

**Example 2**

```
proc sql;
    create table fltrdate as
        select *, min(trgtdist), max(sdobs) from
            (select *
                max(postdose) as maxday from
                (select x.*
                    maxday as minday
                from obsdate as x
                    left join cmed as y
                    on x.prot=y.prot & x.trial=y.trial & x.ptno=y.ptno
                    group by prot, trial, ptno, visit)
                where sdantimc is null & sdobs<=sdantimc & minday>sdantimc
                group by prot, trial, ptno, visit)
                where visit=2 & (postdose>0 & maxday=0)
            group by prot, trial, ptno, visit, trgtdist
            having trgtdist=min(trgtdist) & sdobs=max(sdobs);
quit;
```

The filtering process is achieved by using a series of nested queries. These queries take two passes through the data, first filtering out any data affected
by the concurrent antibacterial rules and then filtering out any unwanted non-posttherapy data.

The innermost query joins the table containing the observation dates created in the steps explained in Example 1 with a table that contains the start day (sdantime) of nonprotocol antibacterial drugs. A left join is used so that patients who have data in the obsddate data set, but not in the cmed data set will still be retained, as well as patients with data in both data sets. A summary variable is then created that contains the earliest study day of each visit for each patient.

The next inner query filters out observations based on concurrent antibacterials. An observation is selected if a patient did not receive an antibacterial drug, or if the observation is before the start of the antibacterial drug, or if all of the data for the visit are after the start of the antibacterial drug. This inner query then creates a summary variable that contains the last study day of each visit for each patient, which is used to determine if there were any posttherapy data for the STFU.

The outer query first filters out any non-posttherapy data for patients who had posttherapy data in the STFU. It then creates two summary variables, one containing the minimum distance from the selection target day (trgtdist) within a visit, and the other containing the maximum study day within each value of trgtdist. The having statement at the end of the query selects the observation within each visit that is closest to the selection target day. If there are multiple observations the same distance from the target day, then the latest observation is selected. At this point, the table fltrdate consists of no more than one observation per visit per patient containing the selected study day for that visit.

Using Subqueries to Select Desired Data

As mentioned previously, the Pathogen Assessment is based on the presence or absence at follow-up of all of the pathogens cultured from that patient at baseline. The first step in performing these assessments is determining the pathogens present in the baseline culture for a particular patient. The follow-up visit cultures are then examined for the presence of those pathogens. If a baseline pathogen is present in the follow-up culture, the pathogen is considered to be persistant and the patient is considered to be a treatment failure. If a baseline pathogen is absent from the follow-up culture, the pathogen is considered to be eradicated. If all of a patient’s baseline pathogens are eradicated at the follow-up visit, the patient is considered to be a treatment responder.

Example 3

```
proc sql;
create table bactdata as
  select distinct a.* from
    (select m.prot, m.trial, m.ptno, m.sdobs, org, d.visit
     from sd.micro as m, bactdate as d
     where m.prot=d.prot & m.trial=d.trial &
     m.ptno=d.ptno & m.sdobs=d.sdobs)
    as a,
    (select m.prot, m.trial, m.ptno, m.org
     from sd.micro as m, bactdate as d
     where m.prot=d.prot & m.trial=d.trial &
     m.ptno=d.ptno & m.sdobs=d.sdobs &
     d.visit=1)
    as b
  where a.prot=b.prot & a.trial=b.trial &
  a.ptno=b.ptno & a.org=b.org
order by prot, trial, ptno, visit, org;
quit;
```

The query selects the baseline and follow-up culture data for all organisms that were present at the baseline visit by joining two subqueries. In the example, the subqueries are separated by blank lines to make it easier to read.

The first subquery selects all of the culture data for the baseline and follow-up visits. The culture data stored in the data set sd.micro are joined with the data set containing the selected visit study days for
the bacteriological data, `bactdate` (which is a final form of `fltrdate` from the previous example). Only the culture data for the specific study days are selected.

The second subquery also joins `sd.micro` and `bactdate`, but it only selects the baseline data (`d.visit = 1`). Therefore, the table resulting from this subquery contains the organisms that were cultured at baseline for each patient.

The results of these two subqueries are then joined. This creates a table (`bactdata`) that contains the baseline and follow-up culture data for all organisms that were present at baseline.

**Combining Data With Coalesce**

It can be determined from `bactdata` if a baseline organism is still present at the follow-up visit. The next step is to add information about whether a baseline organism was eradicated at follow-up (i.e., that the baseline organism was not present in the follow-up culture). Thus, there must be a record for each baseline organism for each follow-up visit (for each patient) with a flag indicating if the organism was still present or was eradicated.

**Example 4**

```sql
proc sql;
create table bactassm as
  select coalesce(a.prot,b.prot) as prot,
         coalesce(a.trial,b.trial) as trial,
         coalesce(a.ptno,b.ptno) as ptno,
         coalesce(a.org,b.org) as org,
         coalesce(a.visit,b.visit) as visit,
         coalesce(a.sdobs,b.sdobs) as sdobs,
         orgexist
  from bactdata as a
  full join
    (select x.prot, x.trial, x.ptno, visit, org, sdobs
     from bactdata as x, baseorg as y
     where x.prot=y.prot & x.trial=y.trial & x.ptno=y.ptno)
    as b
  on a.prot=b.prot & a.trial=b.trial &
     a.ptno=b.ptno & a.org=b.org &
     a.visit=b.visit;
quit;
```

A data set already exists that shows if a patient had a follow-up visit, `bactdate`. This data set has one record per visit per patient, but one record per baseline organism per visit per patient is needed. The subquery accomplishes this by joining `bactdate` with `baseorg`, a data set containing only the baseline organisms for each patient.

This information then needs to be combined with `bactdata`. Since not only observations that are matches between the data sets, but also observations that are in only one of the data sets are to be retained, a full join is used. That makes the use of the `coalesce` function necessary. Coalesce takes the first nonmissing value of the stated variables. For instance, if a particular visit is missing in `bactdata`, but is present in `baseorg`, then the data from `baseorg` are used. If a visit is missing in `baseorg`, but is present in `bactdata`, then the data from `bactdata` are used. If a visit is in both data sets, then the data from `bactdata` are used.

The resulting data set contains one observation per patient per baseline organism per visit. The presence or absence of a baseline organism can be determined from the flag variable `orgexist`, which came from `bactdata`. If `orgexist` is true, then that particular baseline organism was present at that visit. If `orgexist` is false, then that particular baseline organism was absent at that visit. This information is ultimately used to create two assessment data sets, one containing the assessments for all baseline organisms cultured from all patients at baseline, and one containing an overall bacteriological assessment per patient.

**Appending Unlike Data Sets Together**

The query in Example 2 referenced a data set named `cmed` which contained the start days of nonprotocol antibacterial drugs which a patient should not have received during the study. These data come from two data sets: `sd.meds` which contains all nonprotocol drugs taken during the course of the study, and `sd_priorant` which contains all antibacterial drugs taken before the start of the
study. The prior antibacterial drugs were required by protocol to be stopped before the start of the study, but that was not always the case. Those drugs that were continued into the study were also of interest. The earliest study day that a patient started a nonprotocol antibacterial drug which was taken during the course of the study needs to be determined.

A way is needed, then, to combine all prior antibacterial drugs of interest that continued into the study with all of the concurrent antibacterial drugs of interest, and return the earliest occurrence of an antibacterial drug taken during the course of the study for each patient. While the structures of sd.meds and sd.priorant are similar (one record per drug per patient) variables which contain similar data of interest have different names. For example, the variable containing the drug name code is called med in sd.meds and ammed in sd.priorant, and the variable containing the start day of the medication is called sdmstart in sd.meds and sdamstrt in sd.priorant.

Example 5

```sql
proc sql;
create view cmed as
  select distinct prot, trial, ptno,
  sdmstart as sdantimc
  from
  (select *
   from sd.meds
   outer union corresponding
   select prot, trial, ptno, ammed as med,
   sdamstrt as sdmstart
   from sd.priorant
   where sdamstop>1)
  where int(med/10000) in (21,22,23,29) &
  med not in (210400,210500,232000,232900)
  group by prot, trial, ptno
  having sdmstart=min(sdmstart);
quit;
```

Nested within the subquery is a query that selects all observations from sd.priorant for which the drug was continued into the study (sdamstop > 1). At the same time, ammed is renamed as med and sdamstrt is renamed as sdmstart, so that the needed variables share common names with sd.meds. These data are then joined with sd.meds using an outer union corresponding. Joining two data sets with an outer union corresponding is similar to using a SET statement within a data step, the two tables are concatenated.

The outer query then selects from this information observations related to the drugs of interest. A six-digit drug code is used, with the first two digits representing the class of drug. Only certain classes are of interest, and they are selected using the `int(med/10000) in (21,22,23,29)` statement. In these classes, there are some drugs that are considered appropriate for the patient to receive concurrently, which is accounted for by the `med not in` section of the where clause.

The query then selects the record that had the earliest start day of a drug for each patient. The query first sets up a summary variable containing the minimum drug study day start (sdmstart), and then subsets the data using `having sdmstart=min(sdmstart)`.

So Why SQL?

Every example presented here could be performed using data step methods, so why use SQL? Two main reasons: control over data selection and maintainability.

Probably everyone has merged two data sets together, looked at the resulting data and then panicked when the results were not as expected. The culprit quite often is the fact that when both data sets have variables with the same name that were not in the by statement, the values from the second data set overwrite the values in the first data set. This can be avoided using SQL, since you are able to specify which data set a particular variable should come from.

There are also other ways to control how to combine data that are not available in data step methods. Using SQL, data can be combined: without sorting the data sets; using similar variables
with different names; and without using any "by" variables. In other words, data can be combined using SQL in just about any way that can be imagined.

Also, combining data using SQL tends to produce more compact code than data step methods. In cases where disk and memory space are not a problem, why is this an advantage? Primarily because once SQL syntax becomes familiar, it can be easier to read than data step methods, which makes it easier to maintain.

A good example of this is Example 2, which filtered out unwanted observations. This could be done using data step methods, but it would require many steps. First one would have to make certain that obsdate and cmrta are sorted properly in order to merge them together. Then it must be determined if all of the data within a visit occur after the start of the new drug, which would require either finding the minimum study day or using some type of flagging system, both of which would require another separate step or two. Then same type of procedure must be done to check for data collected during therapy, and so on. This results in quite a few separate steps. The query that was used, on the other hand, looks intimidating to novice users of SQL. But once the syntax is learned, the query seems easier to follow, especially since all of the steps required are in one place bracketed by the proc statement and the quit statement, and not spread out like data step statements can be.

Conclusions

SQL can open up many new ways to manipulate data in the SAS System. It allows the combination and manipulation of data in ways that are either not available or are extremely complicated using data step methods.

It does have some drawbacks, though. It can be quite slow and require more computer resources than data step methods. It also can have a steep learning curve for dedicated data step programmers, especially since the syntax used in SQL is completely different from standard data step and proc syntax.

But, when SQL is used appropriately, the advantages of having more control over the data and producing code that is more maintainable than data step methods outweigh the disadvantages of SQL.