DIGITS and DATES: Proc SQL Goes Loopy

Jack Hamilton
Kaiser Foundation Health Plan

Alternate Title

SAS Institute insisted that my original title be changed to

DIGITS and DATES: The SQL Procedure Goes "Loopy"

The new title doesn't have quite the same ring to it, does it?

Overview

- This presentation describes a method to simulate data step looping in SQL.
- Looping is a somewhat specialized requirement; much of what you would do with loops in an algorithmic language is done with set manipulation in SQL, so ordinarily there's no need for this.
- SQL looping is especially useful when the data step is inconvenient or not available.
- In my case (and this is not the case for everyone), I look upon SQL as a challenge to be solved, so I will sometimes use SQL even when the data step is easily available.

A Fundamental Problem With SUGI Papers

This paper is in the Coders Corner section, which is designed for brief descriptions of solutions to particular problems.

Alas, the problem I wanted to solve is too complicated to explain fully in a short paper.

So I'm going to solve a much simpler problem instead. It's almost trivial, so you might think "Why bother?"

The answer is "What I'm showing today is trivial, but it can be scaled up to solve non-trivial problems."

A Simple Problem

The simple problem I'm going to solve today is to list the even numbers 0 through 98 without having an input table containing all those numbers.

The Key To Looping

The key to solving the looping problem in SQL is to have a simple utility data set containing only the digits 0-9.

```
proc sql;
   create table digits
      (digit integer);
   insert into digits
      values(0) values(1) values(2)
      values(3) values(4) values (5)
      values(6) values(7) values(8)
      values(9);
quit;
```

The Key To Looping

By joining the digits table with itself, we can produce any numbers we want. We can subset the results using a WHERE clause.

A Simple Example

```
proc sql;
   select   ones.digit
            + tens.digit*10
               as number
   from     digits as ones,
            digits as tens;
quit;
NOTE: The execution of this query involves
performing one or more Cartesian product
joins that can not be optimized.
```

This produces the numbers 0-99.

A Simple Example

If you look at the resulting data set, you'll see that it's not exactly what you wanted - it contains the right numbers, but in the wrong order. You will see:

```
          0
         10
         20
...
         80
         90
          1
         11
...
```

What Happened?

It turns out that in SAS's implementation of SQL, joins are performed from right to left rather than left to right as you might expect. The fix is simple: just reverse the data sources:

```
proc sql;
   select   ones.digit
            + tens.digit*10
           as number
   from     digits as tens,
            digits as ones;
quit;
```

Obtaining Subsets

If you don't want all the numbers, use a WHERE clause:

```
proc sql;
   select   ones.digit
            + tens.digit*10
              as number
   from     digits as ones,
            digits as tens
   where    mod(calculated number, 2) = 0;
quit;
```

Produces the even numbers 0-98.

Obtaining Subsets

Or change your logic:

```
proc sql;
   select   2 *
            (ones.digit + tens.digit*10)
               as number
   from     digits as tens,
            digits as ones
   where    tens.digit < 5;
quit;
```

Obtaining Offsets

If you don't want to start at 0, you can add an offset

```
proc sql;
   select   100
            + ones.digit
            + tens.digit*10
              as number
   from     digits as ones,
            digits as tens
   where    mod(calculated number, 2) = 0;
quit;
```

Produces the even numbers 100-198.

Remember the Frivolous Law of Arithmetic: almost all numbers are very, very, very large!

Looping With Dates

Looping over a range of dates is just a modification of looping with an offset. Suppose you want the 1000 days starting on January 1, 2005:

```
proc sql;
   select   '01jan2005'd
               + ones.digit
               + tens.digit*10
               + hundreds.digit*100
                  as mydate length=5
format=date9.
   from     digits as hundreds,
            digits as tens,
            digits as ones
   order    by calculated mydate;
quit;
```

A Utility Date Data Set

The Frivolous Theorum of Arithmetic doesn't really apply to dates - you'll pretty much never need to deal with dates outside the range 1886 to 4000 (for our internal standard for unknown date values), so you can put them all into a SAS data set containing all those dates, along with some other useful information, and still not take up much storage space. That lets you do a simple join on dates instead of the Cartesian joins shown above.

A Utility Date Data Set

```
data dates;
   length date 5.;
   do date = '01jan1886'd to '31dec4000'd;
      year = year(date);
      month = month(date);
      dayofmonth = day(date);
      dayofweek = weekday(date);
      julian = juldate7(date);
      lastdayofmonth =
         (date=intnx('month', date, 0, 'e'));
      output;
   end;
   format date date9.;
run;
```

A Utility Date Data Set

See the accompanying paper for some examples using the utility DATES data set.

Why I Learned How To Do This

At my previous employer, we relied on SAS/Intrnet for data collection and reporting.

SAS/Intrnet has two major components - the Application Broker and htmSQL.

The Application Broker uses standard SAS programming - data steps and procedures - and uses the Output Delivery System to create HTML, PDF, or Excel-readable output. It's very powerful, but also complicated.

Why I Learned How To Do This

htmSQL is a combination of HTML and SQL. It's not as powerful as the Application Broker, but it's easier to set up. It can also do master/detail reporting, which the Application Broker cannot easily do. It also gave better control over HTML.

Because of its simplicity and power, I preferred to use htmSQL whenever I could. That meant I had to learn new ways to perform standard tasks.

A Sample htmSQL "program"

```
<HEAD>
<TITLE>htmSQL: Retail
Data</TITLE>
</HEAD>
<BODY>
{query datasrc="retail"}
{sql}
select sales,date,year,month,day
from sashelp.retail
Where year = {&PARAM}
{/sql}
<TABLE BORDER=1>
 <TR> <TH>Sales</TH>
        <TH>Date</TH>
        <TH>Year</TH>
        <TH>Month</TH>
        <TH>Day</TH>
 </TR>
```

```
{eachrow}
 <TR>
 <TD>{&sales}</TD>
 <TD>{&date}</TD>
 <TD>{&year}</TD>
 <TD>{&month}</TD>
 <TD>{&day}</TD>
 </TR>
 {/eachrow}
</TABLE>

{/query}
</BODY>
```

Why I Learned To Do This

The particular application I wanted to write was a file listing - we added report files (PDFs) to a particular directory when they were created, and I wanted the directory page to automatically show the current list - I didn't want someone to have to manual update the index page whenever a new report was added.

I can get the names of the files in a directory using SAS functions, but I needed to put the functions in a loop to get all the filenames. Hence the need for looping in SQL.

The actual code used to obtain the listing is too complicated to explain in a short presentation, so I will describe only simple looping today. The directory listing code is shown in the accompanying paper.

Conclusion

One of the most powerful features in the data step, iteration, can in many cases be simulated in SQL though the use of utility data sets.

Although the examples in this paper are restricted to ordinary programs and htmSQL, the techniques illustrated can be used in any SQL environment, including the SAS Java interfaces and standalone database (Oracle, DB2, etc.) applications.

Acknowledgements

- Richard deVenezia