Paper SD-10
Taking it Home and Putting it into Practice
Diane Cunningham, Southern Company Services, Atlanta, GA

ABSTRACT

We attend SUGI & SESUG to network with fellow SAS® users and hope we'll discover some tidbit of knowledge that can be brought back to the office and put into practice. Well, that's what happened at SUGI 27. I had been working on a large data modeling project when my coworker attended SUGI. When he returned, he shared a paper by Nancy Croonen and Henri Theuwissen, titled "Table Lookup: Techniques beyond the Obvious". The paper contained a discussion of 7 different approaches to table lookups. Future references to this paper will use "the SUGI paper".

The timing of receiving this information was perfect. I read the paper and considered which approach would work best for each of the situations I faced. I ended up choosing two different methods for two different reasons. This paper will discuss each situation and highlight reasons for deciding on each approach.

INTRODUCTION

Georgia Power, a large electric utility, offers certain residential and small commercial customers a price (or rate) called Flat Bill. Flat Bill is a voluntary rate where the customer pays one monthly price for a year regardless of how much electricity they actually use. A unique offer is generated for each individual qualifying customer using their specific electric usage history. For Georgia Power, that's approximately 1.9 million offers. These offers are updated monthly.

In order to ensure that Flat Bill offers are appropriately priced, Georgia Power must predict the customer's monthly energy consumption over the next 12 months. The predicted consumption depends on their usage history for the preceding 1 – 3 years.

However, that history has occurred under varying weather conditions. Since weather is known to be a strong predictor of residential and small commercial energy use, establishing a relationship between the two was necessary to develop a pricing method for Flat Bill offers. The model uses regression to develop an equation that relates energy usage to weather. This equation is then used to predict a customer's usage under normal weather conditions.

Building the dataset to be used in the regression model entails pulling information from several SAS datasets and outside weather data. Only a small part of the total process will be discussed in this paper. All of the table look-up needs of this project were made easier by considering and utilizing the methods described in the SUGI paper.

INPUT DATA SETS

Only active customers that are currently being served on the standard residential or small general service rates with at least 12 months of electric usage history qualify for Flat Bill. The variables required to test for program qualification are located on two separate table structures.

- The account table contains information regarding the customer account including an account number (ky_ba) and status code (cd_ba_st) along with other variables such as customer name, mailing address, etc. The table structure is divided into two SAS datasets; one for residential and one for commercial/industrial. The tables are indexed on account number which uniquely identifies a customer. Each account number only appears once in the input SAS data sets IN1.ACCTRS and IN1.ACCTCI.
- The building table structure contains information regarding the building or premise where electric service is provided. There could be multiple premise records per account. This table includes variables for the rate (cd_tar_s) and operating center (cd_oper_). The operating center is a location variable and is used to map a customer to a weather station. There are seven different weather stations used to model customers in Georgia Power's service territory. This table

is also divided into residential and commercial/industrial data sets. These tables are indexed on three key variables that make a record unique: premise number (ky_prem_), service point (ky_spt), and account number (ky_ba). These data sets are IN1.BLDGRS and IN1.BLDGCI.

Another task performed after the qualifying customers are identified, involved pulling data from a third table

- The billing tables are used to obtain the electric usage history. Usage history is stored monthly and the residential billing table (IN1.BILLRS) and commercial/industrial billing table (IN1.BILLCI) are not indexed. The variables required from this table include account number (ky_ba), service point (ky_spt), premise number (ky_prem_), monthly energy usage (usage), usage month (month), usage year (year), rate schedule (cd_tar_s) and meter read date (mtr_rd_dt). The weather data is matched to the monthly usage by the meter read dates. For the Flat Bill program, usage history is matched to a customer by the account number.

USING THE SUGI PAPER

Croonen and Theuwissen discussed seven different techniques to perform table look-ups. Would these techniques work for the Flat Bill model? The best thing was that the analysis of which methods performed the fastest was already done. The summary table from their paper is shown below:

PROGRAM NUMBER	INDEX ON LOOKUP SAS DATA SET?	REAL TIME (SECONDS)
7-B	YES	0:00:00.40
6-B	YES	0:00:00.53
2-C	YES	0:00:11.66
5-A	YES	0:00:12.02
6-A	NO	0:00:19.78
7-A	NO	0:00:19.88
4-C	NO	0:00:26.15

4-D	YES	0:00:28.53
4-E	YES	0:00:28.81
4-B	NO	0:00:28.91
2-B	NO	0:00:29.14
4-A	NO	0:00:32.18
3-B	YES	0:00:33.06
2-A	NO	0:00:34.11
3-A	NO	0:00:35.10
1-B	YES	0:00:53.25
1-A	NO	0:03:22.10

The first step in the process, identifying all qualifying accounts, required looking at both the account tables and the building tables. Since the account table is indexed on a single variable, this could be used as the look-up table. If we look at the time trial summary for an indexed table, we find that program 7-B is the fastest method. This method uses a SQL INTO clause. It requires using SQL to create a list of account numbers as a macro variable and then selecting the building table records where the account number is in the macro variable list. I quickly found out that there is a limit to the length of macro variables and that 1.9 million 10-digit account numbers exceeds that limit.

The next fastest method (6-B) uses a CALL EXECUTE routine. This is somewhat obscure, but dynamically creates SAS code to use the account numbers from the account table to generate a WHERE clause for the building table. I tried this method and ran into overflow errors, so on to the next fastest (2-C).

This method uses SQL inner join. Inner joins are very efficient when using the index. I tried two variations on the join. In the first example, I put all the WHERE conditions for qualification as part of the SQL where clause. In the second example, I did a very simple join and then used a second data step to complete the qualification tests. The first program took quite a bit longer to run than the second. Although this method worked, I'm less familiar with SQL language so I continued and looked into the next method on the time trial summary.

Example 1

```
proc sql;
   create table cust1 as
      select b.ky_ba,
             b.ky_prem_,
             b.ky_spt,
             b.cd_oper_,
             b.cd_tar_s
      from in1.gprsacct a,
           in1.gprsbldg b
      where b.ky_ba = a.ky_ba and
            a.cd_ba_st = '02' and
            b.cd_spt_s = '02' and
            b.cd_tar_s in ('025','079','028','080') and
            b.cd_spt_t = '0200' and
            b.cd_mpt_t <> 'CST';
   create table cust2 as
      select b.ky_ba,
             b.ky_prem_,
             b.ky_spt,
             b.cd_oper_,
             b.cd_tar_s
      from in1.gpciacct a,
           in1.gpcibldg b
      where b.ky_ba = a.ky_ba and
            a.cd_ba_st = '02' and
            b.cd_spt_s = '02' and
            b.cd_tar_s in ('025','079','028','080') and
            b.cd_spt_t = '0200' and
            b.cd_mpt_t <> 'CST';
quit;
```

Example 2

```
proc sql;
   create table cust1 as
      select a.cd_ba_st,
             b.ky_ba,
             b.ky_prem_,
             b.ky_spt,
             b.cd_oper_,
             b.cd_spt_s,
             b.cd_spt_t,
             b.cd_mpt_t,
             b.cd_tar_s
      from in1.gprsacct a,
           in1.gprsbldg b
      where b.ky_ba = a.ky_ba;
   create table cust2 as
      select a.cd_ba_st,
             b.ky_ba,
             b.ky_prem_,
             b.ky_spt,
             b.cd_oper_,
             b.cd_spt_s,
             b.cd_spt_t,
             b.cd_mpt_t,
             b.cd_tar_s
      from in1.gpciacct a,
```

```
            in1.gpcibldg b
       where b.ky_ba = a.ky_ba;
quit;


data cust3;
   set cust1 cust2;
    where (cd_ba_st = '02' and
           cd_spt_s = '02' and
           cd_tar_s in ('025','079','028','080') and
           cd_spt_t = '0200' and
           cd_mpt_t <> 'CST');
```

This next method (5-A) uses a SET statement with a KEY= option. In this method you use two SET statements. The first SET statement refers to the primary table, or in this case, the building table. The second SET statement is the lookup table, or in this case, the account table (the one that is indexed). Using this method, SET statement commands such as KEEP and WHERE can be used to perform the look-up and the qualifying tests all at the same time. Example 3 shows the programming steps for this method. I decided to go with this method because I liked the succinctness of the programming and did not feel that the logic was too difficult to understand.

Example 3

```
data cust1;  /* Identify qualifying residential accounts */
     set in1.gprsbldg (where=(cd_spt_s = '02' and cd_tar_s in
              ('025','079','028','080') and cd_spt_t = '0200'
              and cd_mpt_t ne 'CST')
              keep = ky_prem_ ky_spt ky_ba cd_oper_ cd_spt_s
                     cd_tar_s cd_spt_t cd_mpt_t);
     set in1.gprsacct (keep = ky_ba cd_ba_st ) key=ky_ba/unique;
     if _iorc_=%sysrc(_dsenom) then delete;
     if cd_ba_st = '02';
data cust2;  /* Identify qualifying commercial accounts */
     set in1.gpcibldg (where=(cd_spt_s = '02' and cd_tar_s in
              ('025','079','028','080') and cd_spt_t = '0200'
              and cd_mpt_t ne 'CST')
              keep = ky_prem_ ky_spt ky_ba cd_oper_ cd_spt_s
                     cd_tar_s cd_spt_t cd_mpt_t);
     set in1.gpciacct (keep = ky_ba cd_ba_st ) key=ky_ba/unique;
     if _iorc_=%sysrc(_dsenom) then delete;
     if cd_ba_st = '02';
```

Once the qualifying list of accounts is created, it needs to be matched to the corresponding billing data. The billing data table is not indexed and neither is the table of accounts just created. Referring to the time trial summary once again, we find that the

fastest way to do a look-up on a non-indexed dataset is the CALL EXECUTE routine and the second fastest method is the SQL INTO clause. For the same reasons mentioned before, these options were not applicable in this situation. The fastest routine that could be applied is the use of the FORMAT procedure and PUT function. The best way I know to describe this method is to think about assigning each of account numbers a value such as *. When applying the format, each of these account numbers now become *. You inherently perform the look-up when you test and keep only the records with a value of *.

In the Flat Bill situation, we not only wanted to pull the billing data for all the account numbers in our list, but also keep the operating center variable (which at this point has been converted to a weather station variable). The weather station variable is used later in the process to merge the billing data and weather data. I was about to put this method aside when I realized it could work due to two important facts. The first fact was that the value assigned in the format does not have to be a fixed value like *. This meant I could use the format to assign each account to its appropriate weather station. Once the format was assigned, I just needed a way to test if an account had been assigned a weather station. That's when I realized the second important fact. The weather station identifier was a 3 character value and the account number was a 10-digit numeric value. If, once the format was applied, the account number still contained a numeric digit, it was not on the qualifying list of accounts and could be deleted. The following shows an example of the final code.

Example 4

```
data custs;
     set cust3;
     start=put(ky_ba,$10.);
     fmtname='$wstn';
     label=stn;
run;

proc format cntlin=custs;
run;

data bill1 (keep=ky_ba ky_prem_ ky_spt cd_tar_s stn usage mtr_rd_dt
                 month year);
     set in1.gprsbill(where=(cd_tar_s in ('025','079','028','080')))
```

```
         in1.gpcibill(where=(cd_tar_s in ('025','079','028','080')));
      if substr(put(put(ky_ba,$10.),$wstn.),1,1) ^in
               ('1','2','3','4','5','6','7','8','9','0',' ');
      stn=put(put(ky_ba,$10.),$wstn.);
run;
```

CONCLUSION

The full Flat Bill Pricing Model takes approximately 4 hours to complete when running alone on our UNIX system using the discussed look-up techniques. It takes approximately one hour to perform the step from Example 4. According to the time trial summary, there is an 84% efficiency gain over a standard merge. That would equate to this one step of the model taking over 5 hours. It's really nice when you can not only learn a new technique at a conference, but bring it home and put it into practice.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Please feel free to contact the author at:

Diane Cunningham
Southern Company Services
241 Ralph McGill Blvd.
Bin 10206
Atlanta, GA 30206
Work Phone: 404-506-3220
Fax: 404-506-3116
Email: dmcunnin@southernco.com