Introduction

This article discusses:

1. How to build a SAS Data Library based on the principles of pseudo-relational Data Base where each SAS Data Set in this library represents a table with data pertaining to a single area of knowledge. One additional SAS Data Set, referred to as 'LINK-TABLE', keeps track of which observations in all other Data Sets are related to each other.

2. How to build queries with direct access to any group of observations from as many tables (Data Sets) as are needed to satisfy given criteria and performing the necessary analysis on the pertinent part of the Data Base in one DATA step.

As an example of this approach, a Computer Performance Evaluation (CPE) analysis system based on data records type 70-75 and 78, collected by Resource Measurement Facility (RMF), was built.

The transformation of the data from the record form where each data element is addressable only by its location (byte number) into a rectangular table (SAS Data Set) where each variable value is addressable by its symbolic name (column name) and row number (observation number) is called, in data base jargon, the normalization process.

If all the records of one type are transformed into one SAS Data Set (table), this table is in the first normal form and carries a great deal of redundant data. In order to eliminate all the redundancy, information from one record type is transformed into various numbers of SAS Data Sets (tables), and a special LINK-TABLE points to all of the associated observations in each table.

Structure of RMF records

All resource utilization measurements are reported by RMF over a span of time that is called the RMF measurement interval. Depending on the type of the resource being measured, either one or a number of records of that type will be created, and this number usually varies from one interval to another. A Standard Header Section , of varying length across record type, and an RMF Product Section are present in every record. The number of other sections varies with the record type, where any section can be repeated various number of times, depending on the interval.

Normalization/Transformation process

Figure 1 shows the transformation of a group of type 70 records - CPU Activity - from several different systems into one SAS Data Set, a table in the first normal form.

<table>
<thead>
<tr>
<th>Standard Header Section</th>
<th>RMF Product Control Section</th>
<th>CPU Data Section</th>
<th>ASID Data Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group of type 70 records from the system SYS1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Group of type 70 records from the system SYS2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Groups of type 70 records from other systems</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SYS1 16FEB88: 00:00:15 INT_TIME CPU Data Section

SYSl 16FEB88: 23:45:15 INT_TIME Obs. for 2nd to (n-1)th int.

SYSl 16FEB88: 02MAR88: 06:30:20 INT_TIME

SYSl 16FEB88: 02MAR88: 18:30:20 INT_TIME

More observations from other systems stored here

Figure 1. Transforming a group of records into one SAS Data Set, creating a table in the first normal form.

In record type 70, the CPU Data section could be repeated up to a maximum of six times (eg., IBM3090/600E). In record type 74, the Device Data section is repeated once per device, resulting in much more redundant data from other sections in the record.
To eliminate this heavy redundancy, the normalization process is carried further. Every RMF record is broken into a number of tables, depending on the number of sections in each record type.

An interval from each system is represented by one observation in the tables below:

<table>
<thead>
<tr>
<th>SYSID</th>
<th>INT_TIME</th>
<th>Product Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYS1</td>
<td>18FEB88:00:00:15</td>
<td></td>
</tr>
<tr>
<td>Obs. for 2nd to (n-1)th int.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYS1</td>
<td>18FEB88:23:45:15</td>
<td></td>
</tr>
<tr>
<td>SYS2</td>
<td>02MAR88:06:30:20</td>
<td></td>
</tr>
<tr>
<td>Obs. for 2nd to (n-1)th int.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SYS2</td>
<td>02MAR88:18:30:20</td>
<td></td>
</tr>
<tr>
<td>More obs. from other systems</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

As a result, each of the tables deals with a single area of knowledge and has no redundant data. Figure 2 depicts the further normalization process for CPU Activity (type 70) records:

An interval from each system is represented by: number of obs. = # CPU Data section repetitions

<table>
<thead>
<tr>
<th>CPUusDS70 Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMF_SID</td>
</tr>
<tr>
<td>SYS1</td>
</tr>
<tr>
<td>SYS1</td>
</tr>
<tr>
<td>SYS1</td>
</tr>
<tr>
<td>SYS1</td>
</tr>
</tbody>
</table>

Figure 2. Type 70 records are transformed into 3 SAS Data Sets.

Processing and Selection of Data

The continuous process of building any Permanent Data Base from new batches of data usually consists of several phases. Validation is usually the first step. Selection of a subset from the validated part of each batch satisfying certain criteria and updating the Permanent Data Base with this subset follows.

To achieve nonredundancy of the data, the CPE analysis system directly transforms batches of RMF record types 70-75 and 78 from a number of different computer systems into 17 tables (SAS Data Sets which are members of one SAS Data Library). Each table contains information from one different section of one RMF record type. There is also a special LINK-TABLE, where one RMF interval is always represented by one observation.

This temporary SAS Data Library of 18 Data Sets is equivalent to a set of tables usually described in Data Base literature as being in the third normal form. The two key variables for the CPE Analysis System, 'SYSTEM ID' and 'RMF Interval Time', are present in every table of the Work Data Base Structure. In the Permanent Data Base, even these key variables (in contrast with a real relational Data Base) will be dropped from the other 17 tables because an additional set of pointer variables will be added to the LINK-TABLE.

Figure 3 lists the set of 18 tables representing the union of three different products (in this case, system architectures). Some of these tables are shared across all three products. The non-shared, non-empty (marked with x) tables uniquely identify a product.

<table>
<thead>
<tr>
<th># Tables</th>
<th>370 XA</th>
<th>XA-3090</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 LINK_TBL</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>2 CPUusDS70</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>3 ASIDDS70</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>4 PAGEDS71</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>5 SWAPPS71</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>6 WORKDS72</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>7 GPRTDS72</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>8 PHYLOG73</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>9 PHYCHN73</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>10 LOGCHN73</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>11 CHPATH73</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>12 DEVIDS74</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>13 PAGSWA75</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>14 IOQDS78</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>15 IOQDS178</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>16 IOQDS378</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>17 IOIOP378</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>18 ALL_DASD</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Figure 3. Table names in CPE System.
The connection between blocks of related observations in all other 17 tables is established via a set of 18 counters created in the LINK-TABLE during the normalization/transformation process. The value of each counter in every observation of the LINK-TABLE is the number of observations in each of the 17 tables for the corresponding RMF interval. The order of the blocks of observations corresponding to the same RMF interval in every table follows the LINK-TABLE order. This is the only set of counters available during the validation and selection phases for direct reading of observations from the other 17 tables (SET statement with POINT option). Therefore, while any of the other 17 tables are being accessed randomly through the LINK-TABLE counters, the LINK-TABLE itself must be processed in a sequential manner only. Figure 4 lists the set of counters in the LINK-TABLE.

The LINK-TABLE plays an active role in the data validation process. For instance, the completeness of the data for one RMF interval can be verified by determining that a specific combination (unique per type of system architecture) of counter variables have nonzero values in the LINK-TABLE observation corresponding to this interval. Even if only one of the values from this combination of counters is zero, then a special utility variable, MASK_PRS (mask the presence of, MASK_PRS='S') is set to a nonzero value. This indicates that blocks of observations in all other tables corresponding to that interval are invalid. Conversely, a zero value of MASK_PRS in this case indicates completeness of the data for that interval.

Another example of data validity testing uses a global variable (duration of RMF measurement interval) present in the LINK-TABLE. The statistical MODE value of this variable per system is calculated and stored in a separate table. Then this value is compared with every interval's value in the LINK-TABLE observation. If the interval's duration significantly deviates from the MODE, then the utility variable (MASK_PRS) is set to another nonzero value (MASK_PRS='M'), indicating that blocks of observations in all other tables corresponding to that interval are invalid. Again, different nonzero values of MASK_PRS designate the various reasons why the data is determined to be invalid or not pertinent.

As the first example of pruning the incoming batch of data, we consider selecting only RMF measurement intervals with CPU utilization of 75% or higher. Program #1 accomplishes both validity checking of the RMF interval duration and selection of the RMF intervals with CPU utilization of 75% or higher in one DATA step. This is shown in Figure 6:

**Example of Program #1 code.**

PROC Univariate finds the most frequently-occurring duration of RMF measurement interval per system and stores it in the UNIV_OUT Data Set.

In the DATA step, the CPUsDS70 table is randomly accessed under sequentially processed LINK-TABLE for calculation of CPU utilization of every interval. Also, the maximum CPU utilization over all intervals for every system is saved in MAX_UTIL Data Set.

**Figure 6. Program #1.**

```
1. PROC UNIVARIATE DATA=LINK_TBL NOPRINT;
2.   BY SYSID NOTSORTED;
3.   VAR I MINUTE;
4.   OUTPUT OUT=UNIV_OUT(RENAME=(SYSID=SMF7_SID)) MODE=INT_MODE;
5. DATA LINK_TBL(KEEP=SYSID INT TIME F_SYSID L_SYSID MASK PRS)
   CPUsDS70(KEEP=SYSID SMF7 INT SMF7_WAT * * * )
   MAX_UTIL(KEEP=SYSID SMF7_MAX_UTIL);
6. RETURN MAX UTIL 0;
7. SET LINK_TBL;
8. IF F_SYSID THEN SET UNIV OUT;
9. IF ABS(INT MODE-1 MINUTE)>3 THEN MASK PRS='M';
10. IF MASK PRS='O' THEN DO;
11.    SUMWAT=0;
12.    DO 2=(CPU70CUH+CPU70CUH+CPU70CUH+CPU70CUH+CPU70CUH+CPU70CUH);
13.    SET CPUsDS70 POINT II;
14.    SUMMARY+SMF7_WAT;
15.    OUTPUT CPUsDS70;
16.    CPU70CUH=II;
17. ELSE DO;
18.    CPU70CUH=SUMWAT/(CPU70CUH+CPU70CUH+CPU70CUH+CPU70CUH+CPU70CUH+CPU70CUH);
19.    IF CPU UTIL LT .75 THEN MASK PRS='I';
20.    IF CPU UTIL MAX UTIL THEN MAX UTIL=CPU UTIL;
21. END;
22. ELSE DO;
23.    CPU70CUH=CPU70CUH+CPU70CUH+CPU70CUH+CPU70CUH+CPU70CUH+CPU70CUH;
24.    IF L_SYSID THEN DO;
25.    OUTPUT MAX UTIL;
26.    MAX_UTIL=0;
27. END;
28. OUTPUT LINK_TBL;
```

**Figure 5. Add two utility vars.**
5. Rebuilds LINK_TBL, CPU\text{DS}70; creates MAX\text{\_}UTIL.
7. Stops the implied DATA step loop execution.
8. If this is the 1st observation of a new system from the UNK-TABLE, then read the MODE value of the duration of RMF interval from UNIV\_OUT.
9. The MODE is compared to the duration of every interval for that system; if the difference > 3 minutes, the interval is marked invalid (MASK\_PRS='M' in LINK-TABLE).
10. Processes only data for valid intervals (all intervals with incomplete data were marked invalid in LINK-TABLE (MASK\_PRS='S') at the normalization step).
12. Repeats the statements that process the entire group of obs. from one interval in CPU\text{DS}70 table that corresponds to the current observation from LINK-TABLE.
19. Any interval with \(<75\%\) CPU utilization is marked nonpertinent (MASK\_PRS='L' in LINK-TABLE).
23. Update the temporary cumulative counter CPU70CUM so that it correctly points to the 1st observation in the group of observations from the next RMF interval.
24,25. Updates counter variable to zero because this group of obs. was never read in from CPU\text{DS}70. It is no longer in the Work Data Base.
26. While processing the last obs. per system, writes max CPU Utilization over all intervals into MAX\_UTIL Data Set.

Intervals with high CPU utilization that perform a large amount of I/O operations are candidates for inclusion in the Permanent Data Base. The main criterion that determines selection of the intervals from each system to be appended to the Permanent Data Base is high Relative I/O, which is defined as the ratio of total nonpaging DASD rate over CPU utilization.

\[
\text{REL} \_\text{IO} = \frac{\text{TOTAL DASD RATE}}{\text{TOTAL PAGING I/O RATE}} \times \text{CPU UTILIZATION}
\]

Program \#2 calculates Relative I/O per interval and stores each system's max Relative I/O in MAXRELIO Data Set, as shown in Figure 7:

\begin{verbatim}
1. DATA LINK_TBL(KEEP=SYSID INT TIME F SYSID L SYSID MASKPRS CPU UTIL)
   DEVIDS74(KEEP=PAGSWA75(KEEP=SMP7 SID SMP7 IST SMP75510))
   RETAIN MAXRELIO 0;
2. SET LINK_TBL;
3. IF F SYSID THEN SET MAX UTIL;
4. IF MASKPRS='O' THEN DO;
5. DO I=DEV74CUM+1 TO (DEV74CUM+DEVIDS4);
6. SET DEVIDS74 POINT=I;
7. DO I=(PAG7SCUM+1) TO (PAG7SCUM+PAGSWA-S);
8. IF MAX UTIL-CPU UTIL1<=0.2 THEN DO;
9. REL IO=(TDADRATE-TPAGRATE)/CPU UTIL1
10. IF REL IO>MAXRELIO THEN MAXRELIO=REL IO;
11. ELSE MASK_PRS='U';
12. END;
13. IF F SYSID THEN DO;
14. OUTPUT MAXRELIO;
15. MAXRELIO=0;
16. END;
17. OUTPUT LINK_TBL;
\end{verbatim}

**Figure 7. Program \#2.**

**Explanation of program \#2 code.**

Numbers below refer to statements in Figure 7.

1. Rebuilds LINK_TBL, DEVIDS74, PAGSWA75; creates MAXRELIO. Funnels variables from the Program Data Vector into whichever Data Sets they belong.
2. For each new SYSID in the LINK-TABLE, reads max CPU utilization.
5-22. Only valid and pertinent (CPU UTIL > = 75\%) RMF intervals are randomly processed from DEVIDS74 and PAGSWA75 Data Sets.
17-19. Relative I/O is calculated only for the intervals with CPU Utilization within the top 20\% of the max CPU Utilization for the particular system. The max Relative I/O per system is saved.
21. All intervals with CPU Utilization below 20\% of the max within each system are masked from future processing with MASK\_PRS='U'.
24,25. Updates counters to process the next valid and pertinent interval in DEVIDS74 and PAGSWA75.
26-27. Counts DEVIDS4 and PAGSWA-S in the LINK-TABLE are set to zero because corresponding observations were not read in and are no longer in the Work Data Base.
29-31. When the last interval per system is processed, max relative I/O is saved in MAXRELIO; its value is reset to zero.

**Building a Permanent Data Base**

The high efficiency of processing the data in the Work Data Base is accomplished by the following:

1. Only the group(s) of observations (direct access with the POINT option of each SET statement) OR group of variables (KEEP option of each SET statement) from the appropriate table(s) participate in the satisfaction of every selection criterion. The choice is based on the following:
• If a table has few variables and very many observations, it is most efficient to rebuild it with fewer obs. satisfying a selection criterion. The Data Set name (KEEP= option) appears in the DATA statement.

• If a table has many variables with fewer observations, then it is more beneficial to use the KEEP option with the SET statement, and only bring into the PDV those variables needed for calculations from that Data Set. No reference of this DATA set name is made in the DATA statement; therefore, it is not recreated with fewer variables in this DATA step.

2. Inappropriate segments of corresponding tables are immediately masked out in the LINK-TABLE and are not read in from the Work Data Base structure at all.

Thus, after going through each validation/selection criterion, the amount of data in the Work Data Base is constantly shrinking.

For example, in Program #1, only CPUsDS70 Data Set is randomly accessed with the exclusion of all invalid intervals; therefore, it is rebuilt without observations for the invalid intervals. At the same time, all intervals with CPU Utilization < 75% are masked out in the LINK-TABLE and never read in again from the other 17 tables.

Program #3 randomly accesses all other 17 tables and finally rebuilds every table in the Work Data Base with the observations for one RMF interval per system with the highest RELATIVE I/O, as shown in Figure 8:

```
%MACRO P_APPEND(DDNAME);
   %LET DNAME1=LINK_TBL; %LET DSNAME2=CpusDS70; %LET DSNAME3=ALl_DASD; %DO 1=1 %TO 18;
      - PROC APPEND BASE=&DDNAME.&&DSNAME&I DATA=&&DSNAME&I;
   %END;
%MEND P_APPEND;

OPTIONS NODSNFERR; 1. DATA LINK TBL(KEEP=SYSID INT TIME SHF7 INT •••)
  DASD CPU UTIL TDAGRATTE TURD RATE IFGRATE •••
  CPUsDS70 ASIDDS70 PAGEDES71 •••
  ALL_DASD NOB_VIRT
  CPUIOPS78(KEEP=SST7 SID SMF7 INT •••)
  ASIDDS70(KEEP=SST7 SID SMF7 INT •••)
  ALL_DASD(KEEP=SST7 SID SMF7 INT •••)
  DDNAME.PNTR_OBS(KEEP=CpusDS70 ASIDDS70 PAGEDES71 SMAPS71 NOB_DASD)

2. IF END LINK THEN OUTPUT DnNAHE.PNTR_OBS;
3. SET LINK TBL END=END LINK;
5-21. IF END PNTR=1 THEN DO;
      • CPUSDS70+1; • IOIOP378+1; ALL_DASD+1;
      • END;
22. ELSE IF N=1 THEN DO;
      • SET DDRARE.PNTR_OBS END=END PNTR; END_PNTR=0;
      • END;
23. IF SYSID THEN SET MAXRELIO;
24. IF MASK PRS='O' AND REL_ID=MAXRELIO THEN DO;
      • OUTPUT LINK_TBL;
      • DO=I+1(CPUIOPS3,J0 TO (CPUIOPS3,J0+CPUIOPS3,J1);
      • SET CPUIOPS3,J0 SPACE=I;
      • OUTPUT CPUIOPS3,J0;
      • END;
35-109. DO=I+1(IOIOP3,J0 TO (IOIOP3,J0+IOIOP3,J1);
            • SET IOIOP3,J0 POINT=I;
            • OUTPUT IOIOP3,J0;
            • END;
112-136. IOIOP3,J0=SUM(IOIOP3,J0,IOIOP3,J1);
137. RUN; SP_APPEND(DDNAME.);
```

Figure 8. Program #3.

Explanation of Program #3 code.

Numbers below refer to statements in Figure 8.

1. Rebuilds LINK-TABLE and all other 17 Tables in the Work Data Base. The KEEP option used with each Data Set name has a complete list of variable names for each Data Set. A special utility Data Set (PNTR_OBS) is referenced in the Permanent Data Base.

2. PNTR_OBS Data Set is updated only once (after all other Data Sets are successfully recreated), just before the implied DATA step loop execution is terminated by SET statement.

4-21. If PNTR_OBS did not exist before, then it is created with all 17 pointers set to the value of 1.
Notice, in addition to the set of 18 counters, a set of pointers (one for each of the 17 tables in the Permanent Data Base) is added to the LINK-TABLE. Each pointer contains the first observation number in the block of observations for each interval in every Data Set, while each previously established counter contains the number of observations in that block. Figure 9 lists the set of pointers in the LINK-TABLE.

<table>
<thead>
<tr>
<th>L</th>
<th>Pointer Variable Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>CPU$0DS70</td>
</tr>
<tr>
<td>N</td>
<td>PHYLOG$073</td>
</tr>
<tr>
<td>K</td>
<td>ASID$0DS70</td>
</tr>
<tr>
<td>T</td>
<td>LOGCHN$73</td>
</tr>
<tr>
<td>X</td>
<td>SWAPPS$73</td>
</tr>
<tr>
<td>B</td>
<td>WORKCSS72</td>
</tr>
<tr>
<td>L</td>
<td>PGRDS72</td>
</tr>
</tbody>
</table>

Figure 9. Pointer names.

The utility Data Set (ddname.PNTR _0 BS) always consists of one observation, which contains a copy of the set of pointers. This copy designates where the 1st observation from the 1st interval of the next upcoming batch of data will start in each table.

Query Building against the Permanent Data Base

In the Permanent Data Base, any group of observations (corresponding to one RMF measurement interval) in any of the 17 tables can truly be directly accessed regardless of the order of processing the LINK-TABLE. Remember, in the Work Data Base, the order of the blocks of observations in each of the 17 tables was dependent on how the LINK-TABLE was ordered.

Construction of a complex query against the Permanent Data Base usually follows these steps:

- Establish which variable(s) from which table(s) are relevant to the query.
- Process those parts of the Data Base, calculate new key variable(s) for this query, and place them in a temporary copy of the LINK-TABLE.
- Now, if necessary, this copy of the LINK-TABLE can be sorted by the query key variables.
- Appropriate blocks of observations from each of the contributing tables will automatically be processed in order of the values of the query key variables, because the corresponding sets of pointers and counters for each of the 17 tables will have been carried along with the sort of these key variables.

An example of one query follows:

MVS provides a function to retain the virtual storage of idle TSO users in processor storage (Logical Swap). Failure to retain the idle address space in the processor storage leads to significant I/O operations to auxiliary storage (Physical Swap). Swap is the state of the Address Space where execution is not allowed. It is important to quantify the amount of processor storage required to achieve a significant percentage of Logical Swap. If no storage is available for MVS use, there will be 0% of Logical Swap (Physical Swap only). Likewise, if there is enough storage to retain all TSO users, there will be 100% Logical Swap (no Physical Swap).

The questions to be answered are:

- What is the shape of the curve between these two points?
- Where on that curve is the optimal point (smallest percentage of idle TSO users kept in storage which gains the largest percentage of Logical Swaps)?

In order to answer these questions, we will extract data from those systems with the number of TSO users \( > 50 \), and calculate, for every interval:

\[
\text{LOGSW\_IN} = \frac{\text{SMF7TOT - SMF7AXD - SMF7LES - SMF7LAX}}{\text{SMF7TOT}}
\]

Then we plot \( \text{LOGSW\_IN} \) versus \( \text{USERS\_IN} \). This query-program #4 is shown in Figure 10.

**DATA LOG SWAP(KEEP=USERS IN LOGSW_IN):**

```
SET DDNAME.LINK_TBL(KEEP=SYSID INT TIME TSO USER SMF7 SAM ASIDDS70 ASIDDS_6 SWAPPS71 SWAPP$_1); IF TSO_USER >= 50;
DO I=ASIDDS70 TO ((ASIDDS70+ASIDDS_0)-1); SET DDNAME.ASIDDS70(KEEP=SHF70ATT) POINT=I;
  USERS_IN=(SMF70ATT/SMF7_SAM)/TSO_USER;
  END;
DO I=SWAPPS71 TO ((SWAPPS71+SWAPPS 1)-1); SET DDNAME.SWAPPS71(KEEP=SW RESON SMF71TOT SMF71AXD SMF71LES SHF71LAX SHF71ESD) POINT=I;
  IF SW RESON=2 THEN DO;
    LOGSW_IN=(SMF71TOT - SMF71AXD - SMF71LES - SMF71LAX)/SMF71TOT;
    I=(SMAPPS71+SWAPPS 1)-1; END;
  END;
OUTPUT;
PROC PLOT;
PLOT LOGSW_IN,USERS_IN;
```

Figure 10. Program #4.

Conclusion

The implemented structure of the SAS Data Library, where each Data Set is a table in the third normal form, saved 50-60% of DASD space versus a library of Data Sets in the first normal form. There was a 60-75% saving in time processing of the average transaction against this structure.

The code for a query of any complexity is very structured, so development of a front end for any type of analysis system could be set up quite easily.

Because of limited space allotment, this article is a very condensed version of the original.