INTRODUCTION

As we enter the 1980's, the importance of information, its organization and accessibility is being realized. This realization is reflected by the preponderance of articles discussing data base management systems (DBMS), Decision Support Systems (DSS) and information centers (IC) in the trade literature. We feel that SAS with its combination of data management, analyses and presentation capabilities is ideally structured to satisfy the need for information management. Thus, we see SAS playing a central role in corporate and governmental information centers.

In order for a DSS to be successful, it must be available to the decision makers. These individuals require a simple, yet powerful, user-friendly interface. Further, for a DSS to be widely utilized in the corporate and governmental setting, it should be able to access large amounts of data efficiently. As we examined SAS with these requirements in mind, we concluded that SAS needed enhancement in two respects.

First, as SAS has grown tremendously in power, it has necessarily increased in complexity. Consequently, the complexity of the documentation has also greatly increased; a fact that may deter the naive or casual user from learning SAS (Bragg, 1980). This would ultimately limit its usefulness in the DSS or IC environment.

In our opinion, the most formidable aspect of SAS to learn involves manipulating and coordinating multiple SAS data sets to create the collection of variables required for an analysis or report. The successful use of the MERGE and SET with their associated SORT procedures, BY statements, and FIRST and LAST variables to extract data from a complex SAS data library requires a considerable degree of sophistication in SAS and in programming techniques. This becomes particularly important in the DSS or IC environment where complex data relationships are the rule rather than the exception. The "what if" questions posed by decision makers often require assembling information from many sources before answers can be given.

The SAS language provides one of the best sets of tools for manipulating data that can be found in a data analysis system. However, these tools are relatively low-level and require a procedural approach. Conversely, it is the nonprocedural approach to data analysis and graphical presentation that is the strength of SAS. We believe that the development of a nonprocedural, user friendly interface to the data management aspects of SAS would greatly increase its utility. It would transfer the power of SAS to a group of important users that currently are relying on programmers for their information processing.

The second area of enhancement involves efficient access to the data stored in a SAS data set. The current method of creating a data base in SAS is to design the data base as if it were a relational data base and employ the SAS data management primitives to effect the operations required by a relational data base (Ingram, 1979). This approach involves using sequential access to the data and is increasingly inefficient as the size of the SAS data sets increases. Thus, although modern data base techniques have been used to good advantage with SAS, they have all employed relatively inefficient sequential data access techniques. In response to user request for more data base functions (see recent SASware ballots), the SAS Institute provided the direct access primitive SET with the POINT= option. However, to make use of this in a data base sense involves building index files and programming by a sophisticated SAS user. The access method becomes more important as SAS moves out of the range of the small-to-medium size, single relation data base into the large-scale, complex, multi-relational data bases commonly found in today's SAS applications.

To address the enhancement of SAS as a tool for the corporate DSS, we have designed DBase.

THE SAS DATA BASE ENHANCEMENT

DBase (Data Base) is intended to provide user-friendly random access facilities for SAS data sets. It allows more efficient access and maintenance of large scale SAS data sets than is currently available in SAS. DBase also provides a human-factored interface to SAS for information retrieval and report generation by the casual SAS user.

DBase has these key features:

- A SAS data set cluster consisting of a Data Area, an Index Area and an Occurrence Area.
PROC DBBUILD, a SAS procedure to support the creation and maintenance of the data set cluster.

PROC DBSEARCH, a SAS procedure to select observations from a SAS data set cluster. PROC DBSEARCH employs random access techniques and query optimization to provide efficient access to a large data set for a complex query.

PROC DBQUERY, a query language preprocessor to translate high level query language commands into SAS code.

THE DBase DATA SET COMPONENTS

The DBase data set cluster is comprised of three SAS data sets: the Data Area, the Occurrence Area and the Index Area (see Figure 1).

The Data Area is a standard SAS data set. It is used as the starting point for the creation of a data set cluster through PROC DBBUILD. The Data Area is unchanged by PROC DBBUILD in the process of data set cluster creation. Thus, the Data Area can still be processed in the normal way by SAS PROCs and DATA steps.

The Data Area SAS data set is comprised of key and non-key variables. Key variables are those that may be used as indices by PROC DBSEARCH to define subsets of the Data Area via random access. Either non-key or key variables may be used to define subsets of the data through sequential access (the normal SAS process of subsetting the data). PROC DBQUERY allows the user to formulate a query without concern for the access type of the variable. PROC DBQUERY will generate efficient SAS code for creating a subset of a large Data Area from a query that employs a mixture of key and non-key variables.

The Occurrence Area is a SAS data set constructed by PROC DBBUILD. The Occurrence Area data set has one observation for each observation in the Data Area. The Occurrence Area serves as a series of multithreaded linked lists that associate records with identical key values.

The Index Area is a SAS data set constructed by PROC DBBUILD. The Index Area is divided into two portions: a series of key variable descriptor records and the actual index records.

The Index Area data set has one observation for each combination of a key variable name and its value. The entries in the Index Area are stored as a series of balanced binary trees that point to records in the Occurrence Area data set.

STORAGE ACCESS PRIMITIVE OPERATIONS

A SAS procedure, DBSEARCH, provides DBase with a set of storage access primitive operations. These operations will create a SAS data set or DBase relation from existing DBase relations using a combination of join variables and select variables. Values for these variables can also be supplied to DBSEARCH in the form of a SAS data set.

PROC DBSEARCH makes selections by assembling a Query Vector which contains, for each key and value pair specified, the starting position of the key variable's linked list in the Occurrence Area, the initial key value and its frequency. This information is assembled after a binary search of the Index Area. The resulting SAS data set or DBase relation is optimally constructed using a "shortest chain first" algorithm.

DBase QUERY LANGUAGE

The DBase Query Language (DBQL) and its preprocessor translator provide the DBase user with a friendly interface to the storage access primitive operations. The query language is modeled after the relational tuple calculus (Date, 1981) using only implicit existential quantification for join variables. Although similar to the QUEL relational language (Stonebraker et al., 1976), DBQL has several significant enhancements. It does not require RANGE statements. In addition to the normal tuple calculus join predicate, an optional MATCHOVER phrase allows a relational algebra-like method for specifying join operations (Chamberlin, 1982). The syntax of DBQL is provided in Figure 2.

Aggregate functions are allowed in both the WITH clause variable list and the WHEN predicates. The DBQL CREATE block can be used anywhere in a SAS program that a SAS data step can be used.

QUERY LANGUAGE COMPOSITION

To illustrate the DBase Query Language's ease of use and conciseness, we express an example query in the standard SAS source language (Figure 3) and in the DBQL (Figure 4). (Note: If the original data sets are large and the number of observations that satisfy the query are few, then subsetting before sorting may be desirable. However, this requires a level of programming of sophistication that probably is not present in the casual SAS user.)

Once the DBQL query has been parsed into an appropriate tree like data structure, a decomposition algorithm (Wong and Youssefi, 1976) will extract a series of one and two relation queries. These subqueries will be passed to the host language code generator.
Figure 1. The Dbase Data Structure

1a. Example Relations:

SUPPLIER
- SNOM
- CITY

SUPPLY
- SNOM
- PARTNUM
- QUANTITY

PARTS
- PARTNUM
- NAME
- SIZE

1b. Logical Relationships for Example:

SUPPLIER

PARTS

SUPPLY

1c. Dbase Structure:

RANDOM ACCESS STRUCTURE

Index Area

Occurrence Area

Data Area

Query

RESULT
Figure 2. DBase Query Language Syntax

CREATE sas-data-set or dbase-relation
ORDERBY order-by-list
WITH variable-list
FROM sas-data-set-list and/or dbase-relation-list
MATCHOVER join-variable-list
WHEN subsetting and joining conjunctive predicate;
printed output options; (e.g., PRINT, MEAN, etc.)

---

Figure 3. Example of Standard SAS Query

Find the names of suppliers in New York who supply size 20 bolts in quantities greater than 200.

PROC SORT DATA=SUPPLIER OUT=TEMPO1;
   BY SNUM;
PROC SORT DATA=SUPPLY OUT=TEMPO2;
   BY SNUM;
DATA TEMPO1; /* This data step would be */
   SET TEMPO1; /* necessary when SNUM */
   BY SNUM; /* duplicates occur in both */
   IF LAST.SNUM; /* data sets */
DATA TEMPO3 (KEEP=PARTNUM SNAME);
MERGE TEMPO1 (IN=IN_ONE) TEMPO2 (IN=IN_TWO);
   BY SNUM;
   IF IN_ONE AND IN_TWO
   AND CITY = 'NEW YORK' AND QUANTITY > 200;
PROC SORT DATA=PARTS OUT=TEMPO4;
   BY PARTNUM;
PROC SORT DATA=TEMPO3;
   BY PARTNUM;
DATA TEMPO3;
   SET TEMPO3; /* Eliminate duplicates */
   BY PARTNUM;
   IF LAST.PARTNUM;
DATA RESULT (KEEP=SNAME);
MERGE TEMPO3 (IN=IN_ONE) TEMPO4 (IN=IN_TWO);
   BY PARTNUM;
   IF IN_ONE AND IN_TWO
   AND NAME = 'BOLT' AND SIZE = 20;

---

Figure 4. Example of DBQL Query

Find the names of suppliers in New York who supply size 20 bolts in quantities greater than 200.

CREATE RESULT
WITH SNAME
FROM SUPPLIER (DB), PARTS (DB), SUPPLY (DB)
WHEN SUPPLIER.CITY = 'NEW YORK'
   AND PARTS.NAME = 'BOLT'
   AND PARTS.SIZE = 20
   AND SUPPLY.SNUM = SUPPLIER.SNUM
   AND SUPPLY.PARTNUM = PARTS.PARTNUM
   AND SUPPLY.QUANTITY > 200;

---
and translated into appropriate storage access primitive operations. For the example query, the two subqueries in Figure 5 would be generated by the query decomposer.

The subqueries are then converted into equivalent storage access primitive operation statements using DBSEARCH procedures. The two subqueries in Figure 5 are translated into the sequence of DBSEARCH procedures that are displayed in Figure 6.

THE PRODUCTIVITY ENHANCEMENT

A nonprocedural interface to SAS for the data manipulation required to satisfy a query can improve the productivity experienced by using SAS. This improvement will occur on the one hand by allowing the query to be expressed in a more concise form. This is an extension of the idea promoted by the "SAS Saves Time" advertising of the SAS Institute.

An even greater improvement in productivity will result from the fact that the DBase Query language provides the unsophisticated SAS user with the capability to formulate complex queries without professional programming support. The nonprocedural nature of the query language requires less training before the user is able to employ the tool effectively. As was pointed out by Martin (1982), the placement of the information management and analyses tools in the hands of the users is the only way to achieve significant productivity gains in data processing.

PERFORMANCE EVALUATION

If SAS is to be truly effective in medium and large data base applications, it must be efficient in its data accessing techniques. We have designed a SAS data set cluster and data access primitive to improve the efficiency of the access currently available in SAS. To evaluate the performance of the design and to compare it to the performance possible with "normal" SAS programming, we assumed that:

- The overall performance is directly related to the number of disk I/O's required to satisfy the query.
- The "normal" SAS programmer would not employ the SET statement with POINT= option.
- There are two types of data bases encountered:
  1) Data base with a known order to the observations (i.e. records arranged in key order).
  2) Data base with an unknown order to the observations.
- SAS minimizes the number of times that it must perform a physical I/O to retrieve the desired observations.

The performance analysis becomes a comparison of disk I/O's versus percent of data base retrieved for each of the data access methods. This comparison is made for each of the types of data bases. The target data base is derived from the example used in Date (1981) and its schema is shown in Figures 1a and 1b. The SAS code for each of the alternatives is outlined in Figures 7 and 8. The solutions (i.e. SAS program code) to the query are not unique. However, we attempted to develop SAS programs that achieved a reasonable degree of efficiency. The programs were designed to minimize the number of observations that would have to be examined to satisfy the query. The results of the analyses are shown in Figures 9a and 9b.

CONCLUSION

This paper presents the results of the extensive design and analysis phase of a project designed to enhance the capabilities of SAS. Our analysis revealed that DBase will significantly increase the efficiency in the access of SAS data sets while decreasing the programming effort. We welcome comments and suggestions during the upcoming development phase.
Figure 5. Example DBQL Query Decomposed Into Subqueries

```
CREATE TEMP
WITH SNUM
FROM PARTS (DB), SUPPLY (DB)
WHEN PARTS.PNAME = 'BOLT'
AND PARTS.SIZE = 20
AND SUPPLY.QUANTITY > 200
AND SUPPLY.PARTNUM = PARTS.PARTNUM;

CREATE RESULT
WITH SNAME
FROM SUPPLIER (DB), TEMP
WHEN SUPPLIER.CITY = 'NEW YORK'
AND TEMP.SNUM = SUPPLIER.SNUM;
```

Figure 6. DBSEARCH Procedures for Example DBQL Query

```
PROC DBSEARCH DATA=PARTS OUT=TEMP01 (KEEP=PARTNUM);
PARMCARDS4;
IF PNAME = 'BOLT' AND SIZE = 20;

PROC SORT; BY PARTNUM;
PROC DBSEARCH DATA=SUPPLY OUT=TEMP02 (KEEP=SNUM)
IN=TEMP01;

KEYS PARTNUM;
PARMCARDS4;
IF QUANTITY > 200;

PROC SORT; BY SNUM;
PROC DBSEARCH DATA=SUPPLIER OUT=RESULT (KEEP=SNAME)
IN=TEMP02;

KEYS SNUM;
PARMCARDS4;
IF CITY = 'NEW YORK';
```
Figure 7. SAS Code To Retrieve The Sample Query From A Large Data Base With An Unknown Order To The Observations

DATA TMP01; SET SUPPLIER;
   IF CITY = 'NEW YORK';
PROC SORT DATA=TMP01; BY SNUM;
DATA TMP01; SET TMP01; BY SNUM;
   IF FIRST.SNUM;
DATA TMP02; SET SUPPLY;
   IF QUANTITY > 200;
PROC SORT DATA=TMP02; BY SNUM;
DATA TMP03; MERGE TMP01 (IN=IN_01) TMP02 (IN=IN_02);
   IF IN_01 AND IN_02;
PROC SORT DATA=TMP03; BY PARTNUM;
DATA TMP04; SET PARTS;
   IF NAME='BOLT' AND SIZE=20;
PROC SORT DATA=TMP04; BY PARTNUM;
DATA TMP04; SET TMP04; BY PARTNUM;
   IF FIRST.PARTNUM;
DATA RESULT; MERGE TMP04 (IN=IN_04) TMP03 (IN=IN_03); BY PARTNUM;
   IF IN_04 AND IN_03;

Figure 8. SAS Code To Retrieve The Sample Query From A Large Data Base With The Observations In Primary Key Order

DATA TMP01; SET SUPPLIER; BY SNUM;
   RETAIN FLAG;
   IF FIRST.SNUM THEN FLAG=0;
   IF CITY = 'NEW YORK' THEN FLAG=1;
   IF LAST.SNUM AND FLAG THEN OUTPUT;
DATA TMP02; MERGE TMP01 (IN=IN_01) SUPPLY(IN=IN_5); BY SNUM;
   IF IN_01 AND IN_5 AND QUANTITY > 200;
PROC SORT DATA=TMP02; BY PARTNUM;
DATA TMP03; SET PARTS; BY PARTNUM;
   RETAIN FLAG;
   IF FIRST.PARTNUM THEN FLAG=0;
   IF NAME='BOLT' AND SIZE=20 THEN FLAG = 1;
   IF LAST.PARTNUM AND FLAG THEN OUTPUT;
DATA RESULT; MERGE TMP03 (IN=IN_03) TMP02 (IN=IN_02); BY PARTNUM;
   IF IN_03 AND IN_02;
Figure 9a. Efficiency Comparison - SAS With and Without DBase

Figure 9b. Efficiency Comparison - SAS With and Without DBase
REFERENCES


For further information contact:

Dr. William Ingram, III
Info Tech, Inc.
P.O. Box 14545
Gainesville, FL 32604
(904)375-7624