DATABASE DESIGN FOR DECISION SUPPORT SYSTEMS
Gregory N. Smith, International Business Machines

ABSTRACT

The Statistical Analysis System (SAS) has powerful capabilities for allowing users to interact with and manipulate data to support decision making activities. Such interactions depend upon the existence of a set of data files (a database) from which data can be selected, combined, and processed to produce information (data useful in a specific context). The structure of that database limits its utility for manipulating data and presenting information. Such limitations can increase programming requirements and restrict a decision maker's understanding of the data. This limits the quality of decisions made.

This paper describes the functions, design goals and structure of a Decision Support System (DSS) database. Then, with this foundation laid, specific descriptions and examples are given for use of SAS to create a database for Decision Support Systems. Employing these suggestions in design and construction of a SAS database will enhance ease-of-use, permit use of the database for divergent needs, clarify reports and graphics, maintain data security and conserve processing power.

FUNCTIONS OF A DSS DATABASE

DSS databases have been developed for a variety of applications. But they are typically used for the following functions:

- Access
  - Selecting relevant information
- Analysis
  - Summarizing information
  - Modeling the user's environment
  - Making "what if" queries of a working model
- Presentation
  - Reports
  - Graphs

INTRODUCTION

A Decision Support System consists of a database of relevant data and the tools necessary to access, analyze and present that data. Such systems are used to enhance knowledge workers' understanding of their environment. A DSS database may be used directly by end-users, or through a programmer intermediary.

The effort spent extracting data from a database adds no value to a user's job. Information does. When the user's "view" of what is required to produce information is complex, the effort required is large. This wasted effort represents lost productivity.

Using SAS to produce information requires less effort than using a conventional programming language. However, SAS usage does require knowledge of SAS logic and syntax, plus programming effort. This complicates the user's "view" of the information available from a SAS database.

This view can be simplified by providing any of the following interfaces between SAS and the end-user:

- Modules consisting of a sequence of SAS statements designed to perform widely used information processing functions. These modules can be selectively combined by users to create programs which produce relevant information.
- A "front-end" processor which specifies information processing options for the user's selection and then generates appropriate SAS code and executes it.
- A facility for transferring information generated by SAS to a software package which the user views as less complex or more familiar.
- Pre-defined reports which users may subset or re-order on-line.

The structure of a DSS database defines the capacity of these interfaces to provide relevant information.
DESIGN GOALS FOR A DSS DATABASE

It is important to remember that a DSS database is used in an environment very different from that of traditional data processing. End-users of a DSS database may have little or no training as programmers. They access a database to increase their understanding of particular situations. The database is a support tool. End-users don’t use this support when the content and format of information obtained is considered less valuable than the effort needed to extract it. Ease of use and output clarity are important concerns.

Most information processing systems generate pre-determined periodic reports. Recipients of these reports sift through them to find applicable information. A DSS database may support this function. However, it must also support unusual requests for information plus reformulation and subsetting of existing reports.

The data files which support traditional information processing systems can be efficiently tailored to providing data for predetermined reports. In contrast, the less structured requests for information from a DSS database require an adaptable data structure. Adaptability is more important than efficiency.

The last design concern is data security. Knowledge workers often use confidential information which must be protected from unauthorized access. Perhaps more important is the need to protect the data from alteration by inexperienced users.

These database design concerns are addressed by the following goals:
- Ease of use
- Adaptability
- Clarity of reports produced
- Data security
- Data processing efficiency

The type of use which a database receives, and the relative importance of each design goal, depends upon the user community which will access that database. The designer(s) of a DSS database should work closely with end-users to ensure that the database will meet their needs.

STRUCTURE OF A DSS DATABASE

TO MEET ITS DESIGN GOALS, THE STRUCTURE OF A DSS DATABASE MUST BE BASED UPON LOGICAL RELATIONSHIPS AMONG THE DATA ELEMENTS, RATHER THAN THAT ARRANGEMENT WHICH MOST EFFICIENTLY SUPPORTS PRE-DETERMINED REPORT FORMATS.

This "relational" format can be represented in a table much like that produced by a PROC PRINT of a SAS dataset. One variable (column) should include a unique value to identify each observation (row) in the dataset. No two observations should be identical and no variable should appear twice in a dataset. If no single variable can uniquely identify each observation, two or more variables may be used in conjunction. The single variable, or set of variables, which uniquely identify each observation are hereafter referred to as the "key" variable.

Although each table, or dataset, cannot have any duplicate values of a "key" variable, many or all of the "key" variable values are duplicated in at least one other dataset. This allows observations in different datasets to be combined or compared using a "BY" statement to match unique "key" variable values in one dataset with matching values, stored as either key or non-key variable values, in another dataset. The collection of datasets which support a decision support system comprise a database.

"Non-key" variables may have repeating values. Such repetition is often the basis for printing a subset of a dataset using a "BY" statement or producing a chart or plot using SAS graphics capabilities.

Once the requirements for building a relational database have been met, the trade-off between including too many or too few variables in a dataset should be based upon whatever makes the user most effective.

End-users who access an overly large dataset may be overwhelmed with excessive data. In addition, duplicating storage of variable values (in several datasets or as repeating values in one dataset) requires extra storage and processing time. However, such duplication can make the database easier to use and provide increased opportunity for subsetting the data. This adaptability permits creation of more, and higher quality, information.
The relationship among variables included in each dataset is an important determinant of the database's utility and processing efficiency. The variables should be related by one central theme. For example, a wholesaler of groceries might maintain a dataset called "yogurt" which contains data on sales of yogurt. This dataset might include the names of buyers, but not their addresses. Address is not specific to yogurt sales, it is determined by buyer name, which probably appears in other product sales datasets.

In this example, address data could be stored in a "buyers" dataset. Address data in the "buyers" dataset could then be matched with yogurt sales data in the "yogurt" dataset by the "key" variable "buyer." Values of variables included in the yogurt sales dataset could be determined by a "key" variable such as "product number." Brand would not be a good "key" variable because most brands offer several yogurt flavors. Brand would not have a unique value for each flavor.

Another determinant of which variable to include in a dataset is whether the variables included would commonly be used together. If not, the theme which relates them may not be specific enough.

In a database designed for decision support activities, if more than one variable can be used as a "key" to uniquely identify each observation, all such variables should be included. This practice increases storage requirements, but facilitates ease-of-use. Matching datasets may involve merging with a third dataset just to get the variable which allows merging by unique observations. Providing datasets with supplemental "key" variables may simplify pathways between datasets. Therefore, end-users need to understand less of the database structure. Making this structure transparent to users is a focus of the relational model.

DATA SECURITY

Data security must also be considered when designing a DSS database. Sensitive data can be confined to one or more datasets which may be restricted by password access (refer to following section on SAS functions). Another technique for restricting data access is to store confidential datasets on a separate disk of a direct access storage device (when using the VM operating system). Only those users with authority to view this dataset need have access to that disk.

A separate aspect of maintaining data security is data integrity. All authorized users who access the same set of variables should get the same variable values. If variables duplicated in a second dataset are subject to change, they might be retained as outdated data after a change is made to the first dataset. This means that "key" variables should not be subject to change. If variables subject to change were confined to one dataset, only that dataset would need to be changed during update. Datasets NOT subject to change should be protected by a read-only password.

SUMMARY DATASETS

Upper level knowledge workers generally require only summarizations of the data used by their subordinates. In fact, too much detail clouds the "big picture." The needs of these users might best be met by a dataset of summary information.

Summary datasets can store the results of calculations performed on a dataset. They may be required to produce reports and graphs. They may offer some security protection when used to summarize data which users should not have access to, such as the names of respondents to an opinion survey. So why not remove the burden of creating summary data sets from database users by providing them as supplements to the more detailed database?

One valid response is "excessive database storage requirements." Another is "use of summary datasets limits the available possibilities for combining and analyzing data." However, if users do not use a database because it is overwhelmingly detailed or the steps necessary for summarizing that detail appear prohibitive, the database will not adequately support their needs. Provision of summary datasets may be the only way to support such users.

SAS FUNCTIONS FOR DATABASE CREATION

USING A PERMANENT SAS DATABASE
The Statistical Analysis System (SAS) has very powerful capabilities for database administration. SAS can automatically allocate and manage space for a database and display the contents and format of that database. The bulk of these capabilities are invoked when users specify a two level name, such as "SALES.SOAP", to create and use a permanent SAS database. Use of a permanent SAS database reduces processing time for SAS jobs because data need not be input for each program, it is already stored and formatted. This also removes a level of complexity from the users interaction with data.

Permanent SAS datasets can also provide security checks on use and alteration of data. Use of the "PROTECT=password" option on the DATA statement protects the dataset from alteration.

```
DATA SALES.CEREAL(PROTECT=SHHP);
INPUT PRODID TYPE $ BRAND $ SALES BUYER $;
```

The dataset can be read without specifying the password, but it can’t be altered. The password must be specified to allow use of PROC EDITOR, PROC SORT, PROC RANK or PROC APPEND.

```
PROC SORT;
DATA=SALES.CEREAL(PROTECT=SHHP);
BY TYPE;
```

The DATA statement option "READ=password" can be used in the same manner. This option protects the dataset from being written to or read unless the password is specified. This "READ=password" option should be used for protecting confidential datasets from unauthorized access. Both password options can also be invoked, after creation of the dataset, by using the PROC DATASETS procedure.

```
PROC DATASETS DDNAME=SALES;
MODIFY BUTTER(READ=SHHR);
```

The contents of a database, format of its variables and any variable labels the user might specify can be displayed using the PROC CONTENTS procedure.

```
PROC CONTENTS DATA=SALES. ALL;
```

The listing produced is a useful reference for writing programs which use the permanent SAS datasets for decision support activities.

**UNIMIZING PROCESSING TIME REQUIREMENTS**

Intelligent use of certain SAS techniques can minimize the requirement for processing time during both database creation and use.

Correction of erroneous data and updating of datasets can be performed using PROC EDITOR. This procedure eliminates the need for executing a SAS job and provides a rapid facility for making changes to a dataset. It is very useful for datasets which are continually updated. The drawback is that once changes are made, the dataset cannot be recreated. It is therefore advisable to store a backup copy of the dataset or the program used to create it.

Future processing time requirements can be reduced by planning an efficient dataset sorting strategy. Processing time increases at a decreasing rate as more variables are specified in the BY statement used with a PROC SORT procedure. In some cases, performing all the required sorts in one PROC SORT statement accomplishes the same end as performing multiple PROC SORT statements.

```
PROC SORT;
BY TYPE BRAND PRODID;
```

This saves processing time. If data is commonly accessed in a certain sorted order, the dataset should be stored in that order.

**PROVIDING DESCRIPTIVE LABELS**

Significant steps can be taken when creating a database to enhance output clarity. SAS requires that variable names be 8 characters or less. However, if a variable can’t be described in 8 or less characters, the database creator can assign a label of up to 40 characters to the variable by using the LABEL statement. This label is stored with the permanent SAS dataset and is printed out by certain SAS procedures when that dataset is used. It should be noted that some reports and graphs might not have room to display a long label and will concatenate it to fit space requirements. Printout of labels on SAS/GRAPH* is restricted to 16 characters or less.

```
DATA SALES.CEREAL;
LABEL PRODID=PRODUCT ID NO.;
INPUT PRODID TYPE $ BRAND $ SALES BUYER $;
```

* Trademark of Statistical Analysis System Institute, Inc.
Variable values can also be assigned labels by using the PROC FORMAT statement with either the VALUE or PICTURE options. These labels make results more understandable. They are stored in a permanent SAS database and alter values for printout only.

The VALUE option can be used when detailed information is not useful. Numbers can be printed out as words (0=no, 1=yes), character values can be expanded for clarity of definition (y=yes, n=no) and ranges of values can be summarized (1-10=low).

    PROC FORMAT;
    VALUE Resp 1=YES
              0=NO;

The PICTURE option allows variable values to be displayed using a SAS format or a format which the database designer specifies. For example, the meaning of a cost variable could be clarified by assigning it a DOLLAR format. This would display the variable values with 2 digits to the right of the decimal, commas, and a dollar sign.

    PROC FORMAT;
    PICTURE Cost Revenue='99,999,999.99'
                 (PREFIX='$');

VALUE and PICTURE Formats are used for information display when a FORMAT statement is included in a PROC PRINT procedure.

    PROC PRINT;
    FORMAT Revenue Cost .;

RECODING VARIABLES

Variable names may be input and arranged in a temporary dataset in a form which is easy for the database creator to use; but they should be displayed in a form which is understandable to decision makers. Variables can be renamed (up to 8 characters long) using the RENAME statement, either in the DATA step

    DATA Sales.Yogurt;
    Set Temp2;
    RENAME OldName=NewName;

or as an option on the DATA statement.

    DATA Sales.Cereal
       (RENAME=(OldName=NewName));

It is sometimes necessary for a dataset to be transposed so that several variables are combined into one variable. This situation can occur when responses to a related group of survey questions are stored as variables and when several variables comprise an array.

Transposing a dataset is useful for presenting variables as parts of a whole. In addition, transposing data is often necessary for creating charts and plots. Transposing a dataset can be accomplished by using PROC TRANSPOSE.

    proc transpose out=Temp4;
      by Cost Revenue;
    run;

or by redefining several variable names as values of one new variable and setting the values of those old variables to values of a second new variable.

    sales=sale1; product=szap; output;
    sales=sale2; product=yogurt; output;
    sales=sale3; product=bologna; output;

It is not reasonable to expect end-users of a DSS database to understand and apply the complex logic needed for transposing a dataset. If a general need for a transpose operation can be identified, this operation should be performed and output as a summary dataset.

DESIGN OF DESCRIPTIVE MNEMONIC LABELS

Mnemonic labels provide a terse description of datasets and computer programs used to create and use those datasets. However, the standard operating system and SAS limitation that program names and data sets be described in 8 characters or less limit the descriptive ability of the labels. These descriptions can be enhanced by dividing the 8 characters into sections which describe different characteristics of the program or dataset.

Let’s return to the example of a dataset for Yogurt sales. This dataset could reside in a "SALES" database (SALES.YOGURT). The name of the program used to create this dataset should indicate what the program does and on what data. It could be called DYOUGURT. "D" refers to database creation. Other useful prefix labels include "S" for creation of a summary dataset, "P" for creation of a report and "G" for creation of a graph. If you need to write more than one program which uses the same dataset for a similar purpose,
you could use numbers or other descriptive letters (GYGURT2).

This mnemonic label has 3 fields (PXXXXXXXX). The "P" field refers to purpose of the program. The "XXXXXXXX" field describes the data used. The "N" field describes the iteration of a program of purpose "P" and content "XXXXXXXX."

Many labeling strategies could be used to describe a program or dataset. The key is that the one selected should display the most descriptive identifiers and be used consistently throughout the database.

The value of a good mnemonic labeling scheme increases when documentation on database content and architecture is inadequate. Database creation and construction is NOT COMPLETE until documentation is prepared on the contents of all datasets, the keys necessary to connect datasets and procedures necessary for recreating the database. However, many database designers and programmers view documenting their work as an unpleasant afterthought. In such cases, the only guides to using the database may be the dataset and program labels plus a "PROC CONTENTS" listing of the database.

**CONCLUSION**

A DSS database is used by decision makers for accessing, analyzing and presenting relevant information. It is most useful when it is designed to address the diverse and changing needs of its users. This can be accomplished by use of a relational database architecture, security protection features and summary datasets. In addition, SAS functions should be employed for database library management, minimalization of processing requirements and provision of descriptive labels. Implementation of the techniques and database architecture described in this paper will result in a database which can support current and future information needs of decision makers.

**REFERENCES**
