Throughout history one principle seems to emerge: Great ideas are simple ideas. This is particularly true in data management. All too often, data-base management systems run the danger of becoming cumbersome, inflexible, and problematic. The logical interconnections tend to multiply as new variables and types of data queries are added. [1]

We must try to describe the data in a way that will avoid the entanglements that can build up in complex data structures. The desired data description techniques will (1) be understood easily by users with no training in programming, (2) make it possible to add to the database without changing the existing logical structure of application programs, and (3) permit the maximum flexibility in formulating unanticipated inquiries. Hierarchial, tree, or pointer-linked logical data structures can often inhibit changes that may be needed as the database grows. [1,2,3]

The most natural way to represent data is with a data matrix, also known as a two dimensional table, or flat file. In this form of data representation, the columns are the variables, or data item types, and the rows represent observations. The most complex tree or hierarchical data structure can be reduced to a collection of flat files with some redundancy. The tables, or flat files, are rectangular arrays that have the following properties:

- Each column in a table represents a single valued variable. Thus there are no repeating groups within the observation. For example, an employee information record would not have a column that might contain multiple values, such as educational institutions attended.
- All rows are distinct, with no duplicates.
- Tables are column homogeneous; that is, in any column all items are of the same kind.
- Each column is distinct, with a unique name assigned.
- Both rows and columns may be viewed in any sequence with out affecting the information content of the table. [2,3,4]

As with most "new" concepts in the data processing field, the enthusiasts of the relational data management approach have embellished the subject with their own set of jargon. (See Fig. 1) The table is referred to as a relation. The columns are known as attributes. Thus a table with n columns is an n-tuple relation, or a relation of degree n. All tuples of a relation are defined by the variable name or attribute.

The domain contains the range from which a value for an attribute, or variable, can be selected. The number of values in a domain can be extremely large or very restricted. Each tuple must have a key, or ruling part, which identifies it uniquely. The ruling part can be composed of more than one attribute. The remaining attributes, the dependent part, describe the object defined by the relation. The ruling part should not contain any redundant attributes. Any attribute of the ruling part which, if removed still leaves the ruling part unique, should be placed in a different relation or in its own separate relation.

Thus, a relational database is formed by assembling collections of tuples into relations. One follows in assembling the relations is called normalization. This process must be followed so that any logical representation of data can be reduced to a collection of flat files, or relations. This allows the tables to be constructed so that: (1) no information about the logical relationships between data items is lost, and (2) set theory manipulations may be performed on the data. [2,4,5]

Normalization proceeds in three steps: [1,2] This process requires that the domains for each attribute permits only simple values. Thus an attribute value can not itself be a relation. This restriction disallows nesting a repeating data element or group of elements. If a nested group is small it may be represented by a fixed number of attributes assigned to different tuples over the same domain. Otherwise the repeating group must be established as a separate relation.

Once all the nested groups have been removed, the resulting relations are said to be in first-normal form. If the value of one attribute A is always determined by the value of another attribute B, then A is functionally dependent on B. This is the case with the ruling and dependent parts of the relation.

To avoid another level of redundancy, the functional dependency of the attribute is considered. Whenever an attribute of the dependent past is functionally dependent on a subset of the attributes of the ruling part, the attribute should be moved to a separate relation or added to another appropriate relation. The removal of attributes which display functional dependencies on
The comparison of an SAS data set to the anatomy of a relation in third normal form:
1) The SAS data set PAT-HEAD is the relation of degree 6, called PAT-HEAD.
2) Tuple with an attribute, or name, PAT_NAME
3) Observation or row of the relation
4) HOSP_NO is the ruling part of the relation

This series of transformations help to reduce redundancy. It also serves to make explicit the functional dependencies which remain and prepare the data to allow set manipulations.

The key to flexibility in a relational database lies in the ease that one can manipulate the relation to form new ones to respond to inquiries. The necessary data manipulation operators includes union, intersection, difference, projection and join. (See Fig. 2)

Union combines two relations such that any tuple present in either relation is present once in the resulting relation. The intersection of two relations is such that any tuple present in either relation is present once in the resulting relation.
relations will eliminate all but the tuples common to both relations. The difference of two relations removes from the first relation all those tuples which exist also in the second relation. Projection reduces the relation by limiting the domains. The join operation forms a new relation which contains the data elements of two relations. An uncontrolled join is equivalent to a cartesian product. Joins may be controlled by computing data element values in matching domains. The most common of the controlled joins is the equijoin. To form the product of an equijoin, the equivalent entries in the matching domains are selected and all possible concatenation of tuple-pairs are created. (See Fig. 2)

The notation discussed above describes the operators of a relational algebra. A relational algebra operation is one which takes one or more operations as its operand(s) and manipulates them to form a new relation. In order to arrive at a desired relation, the user must specify a sequence of relational algebra operations. An alternative to giving the user access to the relational algebra operators is to provide a more automatic approach, the relational calculus. With this method, the user defines the desired result and lets the system decide what operations are necessary to extract the result from the database. Relational calculus is a notation for defining a relation which is to be derived from the existing relations in the database. Thus there are three levels of automation possible with relational database management systems: one tuple at a time, algebra and calculus. SAS provides a vehicle for applying relational database techniques to your data management needs. The definitions of an SAS data set correspond to the requirements for relations discussed above. The process of normalization must be performed by the application programmer during the design process. The following section takes each of the relational algebra primitives and gives an example of their implementation in SAS. The example is drawn from the Tumor Registry System described elsewhere.

Union: The union is accomplished by the SET command. (See Fig. 4)

Intersection: This function is achieved in one pass of the data by use of the SET MERGE and IN commands. (See Fig. 5)

Difference: (See Fig. 6)
### RELATION ALL: UNION OF ALIVE & DECEASED

<table>
<thead>
<tr>
<th>HOSP_NO</th>
<th>BIRTHDATE</th>
<th>SEX</th>
<th>RACE</th>
<th>EXP_DATE</th>
<th>PAT_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01SEP45</td>
<td>M</td>
<td>W</td>
<td></td>
<td>JOHN DOE</td>
</tr>
<tr>
<td>2</td>
<td>02OCT46</td>
<td>F</td>
<td>W</td>
<td>10OCT79</td>
<td>MARY DOE</td>
</tr>
<tr>
<td>3</td>
<td>03DEC47</td>
<td>N</td>
<td>B</td>
<td>01AUG78</td>
<td>JIM EVERS</td>
</tr>
<tr>
<td>4</td>
<td>04JAN48</td>
<td>F</td>
<td>B</td>
<td></td>
<td>LOU EDAW</td>
</tr>
</tbody>
</table>

#### Projection: The projection is accomplished by use of the KEEP command. (See Fig. 7) A notable difference is that the projection operator also specifies the order of the variables in the new relation, while use of KEEP retains the order found in the original relations. This may be important if the relation is to be passed to PROC MATRIX where the program is dependent on the column order of data. In that case it is wise to use the KEEP feature of the PROC MATRIX FETCH command.

#### DATA JOIN: MERGE ALIVE (IN=IN ALIVE) PAT SITE (IN=IN SITE); BY HOSP_NO;

### RELATION ALL2: DIFFERENCE OF ALL & DECEASED

<table>
<thead>
<tr>
<th>HOSP_NO</th>
<th>BIRTHDATE</th>
<th>SEX</th>
<th>RACE</th>
<th>EXP_DATE</th>
<th>PAT_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01SEP45</td>
<td>M</td>
<td>W</td>
<td></td>
<td>JOHN DOE</td>
</tr>
<tr>
<td>2</td>
<td>02OCT46</td>
<td>F</td>
<td>W</td>
<td>10OCT79</td>
<td>MARY DOE</td>
</tr>
<tr>
<td>3</td>
<td>03DEC47</td>
<td>N</td>
<td>B</td>
<td>01AUG78</td>
<td>JIM EVERS</td>
</tr>
<tr>
<td>4</td>
<td>04JAN48</td>
<td>F</td>
<td>B</td>
<td></td>
<td>LOU EDAW</td>
</tr>
</tbody>
</table>

**Fig. 4**
The relation ALL (all patients) resulting from a UNION operation.

**Fig. 5**
The relation LIVEFMLE (all living female patients) resulting from the INTERSECTION operator.

### RELATION PROJECTN: PROJECTION OF PAT_HEAD

<table>
<thead>
<tr>
<th>HOSP_NO</th>
<th>PAT_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>JOHN DOE</td>
</tr>
<tr>
<td>2</td>
<td>MARY DOE</td>
</tr>
<tr>
<td>3</td>
<td>JIM EVERS</td>
</tr>
<tr>
<td>4</td>
<td>LOU EDAW</td>
</tr>
</tbody>
</table>

**Fig. 7**
The relation PROJECTN resulting from PROJECTION operator.

---

In some respects, SAS is more general than other relational database systems. SAS will allow any of the set operations to be performed on relations with disjoint domains. In this case, SAS will automatically supply null, or missing values, for the appropriate fields. Also SAS does not automatically select n-tuples or observations based on their ruling part attribute values. Thus, the need for the IN, BY, and subsetting IF statements to correctly perform any of the subsetting statements (ie: intersection, difference, join).

Since SAS is processing the data in a sequential method, it is left to the application to determine that the data is in the proper order. Therefore, one must usually sort the data set, using PROC SORT, prior to performing one of the set operations.

An extension of the relational database management facilities found in SAS can be implemented using the programming statements. These can create more complex logical selections in concert with the join, union, intersection or difference operations. The set operations defined above can also be combined in SAS for example a projection, KEEP statement, is easily combined with any of the other operations.

When discussing database techniques it is useful to also consider topics of data independence and the degree of binding of the database structure. As long as the applications are contained within the frame work of SAS, the degree of data independence is high. The application is never concerned with the physical storage method of SAS and is rarely concerned with the data type be-
The relation JOIN resulting from the EQUIJOIN operator (all living patients and their specific case data).

The relation JOIN2 resulting from an unconditional JOIN. HOSP NO's 2 and 3 were present in PAT SITE but not in ALIVE. In a strict relational database interpretation of the JOIN operator they should not appear in relation ALIVE2.
boundaries, such as annual, active or inactive, of alive or deceased. This solution requires that the person accessing the data have more detailed knowledge of the structure of the database, and thus should be viewed with caution. Many aspects of this problem have been addressed in a SAS technical report.[8] Another negative aspect of using relational database techniques in SAS lies in the level of automation possible. As has been seen, the implementation of relational operations in SAS lies somewhere between the one tuple at a time and the relational algebra level of automation. The operators are easily implemented in SAS and straightforward once the approach is thought out. However, in other than relatively simple applications, they often require an experienced SAS user or programmer. This reduces the utility of SAS among management personnel and restricts the ability to distribute the data processing power to the actual user.

An obvious solution to this would be through the use of MACRO's. The use of MACROS to formulate a higher level special purpose query language from a lower level language is a well documented approach.[9] In particular it has been implemented with some success in data base management systems.[10, 11] The minimum requirements for this are a MACRO library facility and the ability to pass argument through the MACRO's. Currently SAS, supports neither a MACRO library or argument passing facility. These two features are essential if one is to create generalized MACROS that can be used as a query language for inexperienced SAS users.

In summary, although SAS is not a DBMS, its organization permits and encourages relational DBMS techniques in managing data. The advantages accrued are: ease of use, flexibility, torespond to information documents, selectibility of data, data independence and clarity of the data structures. The disadvantages lie in the resources SAS uses when one employs SAS to manage large amounts of data, and in the level of knowledge of SAS required to perform complex manipulations of the data.

REFERENCES
5) Codd, E.F. 1970. "A Relational Model for Large Shared Data Banks".
11) Software Ag of North America. 1978. ADARAS system overview.