A Set-Theoretic View of Database Representation

Henry Beitz Ann Arbor Michigan

To achieve what I believe to be an acceptable Information Management System, I propose that a set-theoretic view of a database be adopted. Most potential database users use the language of set theory in their day to day references to the objects and relationships of their databases. Instead of twisting these natural language references to ‘collections’, a representation that is modelled on sets should be implemented. Among the many advantages to be gained by adopting this approach are: non-redundant storage of data common to two or more application environments, controlled access to the entire database by any data-dependent key, rapid selection and isolation of easily specified subsets of data, and the extraction of only those data necessary to satisfy specific needs.

This description of database representation will include some conceptual aspects and will address specific problems related to the computer-based database. Some of the areas that will be addressed are security, integrity, accessibility, and flexibility.

Initially some premises about databases must be stated: they are essentially a repository for data and for the relationships pertaining among data; the data may be either static or dynamic and provision must be made for adding, removing and altering data; access to the data in a timely, unrestricted and also an unpredictable manner is desirable.

All databases are really representations of relationships, (or relations). A relation names a specific connection between two things. In this sense an attribute name is a relation that is given to the connection between an object and the **'value'** of the relation with respect to that object. Suppose that one of the many objects represented in a database is a product produced by a company. For this product to be inserted into the database we must store an encoded representation of at least one property that is ascribed to this product. Let us assume that we want this product to be known as PRODUCT-XYZ. No matter how this piece of data is recognized it has to have a name. We will call it the ITEM-NAME (i.e., the value of the relation ITEM-NAME with respect to our product is PRODUCT-XYZ).

There are numerous methods of representing this **property**, as we will call it, of the product. The complete string 'ITEM-NAME = PRODUCT-XYZ' is one method and just the string 'PRODUCT-XYZ' in a very specific place in the record which represents the product is another. Positional significance is so much a part of our *a priori* experience that it is often easy to lose sight of its tacit meaning. For example, almost everyone would make the following simple association:

$$\mathbf{236 = (2 \times 100) + (3 \times 10) + (6 \times 1)}$$

In much the same manner card-columns and fields, not to mention data descriptions and schema, are very much an integral part of our data-processing environment.

There are other relationships, which require very much more explicit representation. These are relationships, which must be considered as data. For example, if two products must be sold together and are therefore mutually dependent, this dependency needs to be recorded in our database. Another example would be the relationship between a component and the assembly of which it is a part. As systems become more sophisticated, there is an emerging tendency for this latter type of relation to dominate the database. This would seem to be a logical outcome of integrating complex sub-systems. Since it implies non-redundant storage of commonly used data, it is desirable. This in turn ensures that any user examining a dynamic datum sees the most recent value of that datum.

So much for relations. It goes without saying that the ability to add, delete and update representations of objects must be provided. There are many ways of approaching updating. However, we would like to suggest that a process analogous to making a course change while flying under *Instrument Flight Rules* conditions be used. Any partial updates or updates achieved by over-writing in place must be avoided. The present course must be maintained and at an extrapolated time and place having determined the new heading, ground speed, etc., the course alteration is made. This will not only guarantee the integrity of the database, but will also permit recovery without loss of data in the event that a system failure occurs.

The most significant reason for proposing this particular approach to an Information Management System is that all too often the premises that influence an IMS design become obsolete shortly after the implementation has been completed. The proposed system will allow an organization to dynamically reorganize their database without restricting access to the system. The IMS design may truthfully be described as a solution to the **unpredictable inquiry problem**. Database reorganization to enhance response time for a specific user performing an experiment or responding to a crisis will be possible as part of the proposed IMS.

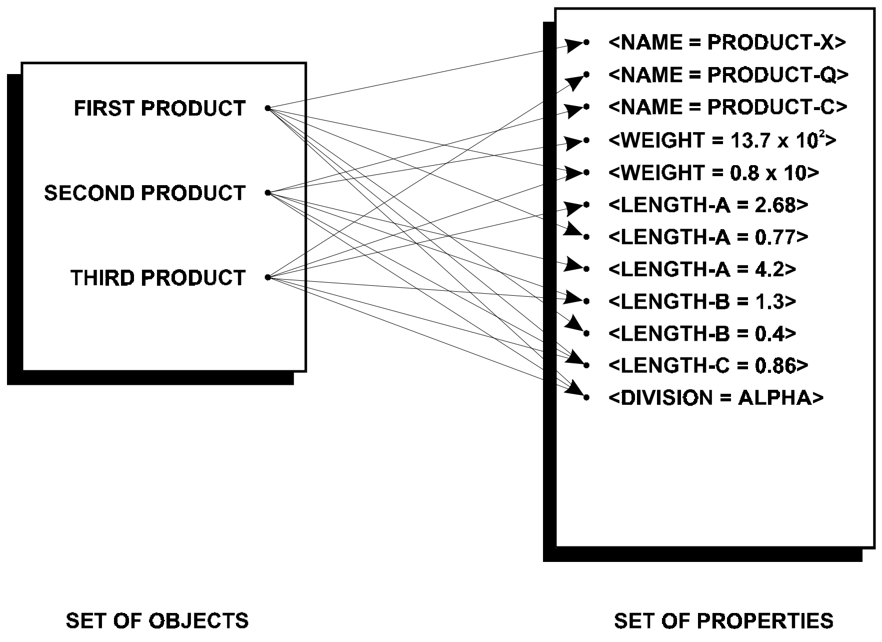

FIGURE 1. A SAMPLE DATABASE

Every object about which data will be recorded really has an infinite number of properties. In a computer-based system, we must confine our representation to the comparatively small number of properties relevant to our needs. The IMS may be conceived as a mapping from a set of objects onto a set of properties as shown in Figure 1:

Neither of the enclosed sets is really the database. In fact, the database is really nothing more than the arrows in Figure 1. The set of properties is data, but that is all. By itself, the set of properties is almost meaningless. The set of objects serves as nothing more than a rallying point for collections of properties. The set of objects does not even contain the association that a user makes to identify a specific product. In fact, all access to the database can only be made by way of the set of properties.

If, for example, ***the WEIGHT of the third product in Figure 1*** is needed, a user would have to start by invoking the inverse mapping and would have to know that NAME = PRODUCT-Q. The inverse mapping would identify the correct rallying point from which it will be possible to find WEIGHT = 0.8 x 10.

Another example might be ***find the largest dimension of all products with WEIGHT = 0.8 x 10.*** Again, we start off in the set of properties. This time, the inverse produces two rallying points and the following result is produced quite simply:

PRODUCT-X with largest dimension LENGTH-C = 0.86
PRODUCT-Q with largest dimension LENGTH-A = 2.68

Most IMS functions can be made up from a sequence of two types of basic functions. The two basic kinds of function are **selection** and **extraction**. Selection is often enough to satisfy an inquiry in its own right. An example using the simple database of Figure I would be, ***are there any objects having the property LENGTH-A < .5?*** A slightly more complex inquiry requiring a more explicit response than simply yes or no would be ***how many objects have LENGTH-B > 0.2?*** (From the diagram, the response to the former example will be seen to be ***no*** and the response to the latter example will be ***3***).

The selection criteria could become considerably more complex. For example, ***how many of the products with***

WEIGHT<10^2 also have a LENGTH-A < 1? (The response should be **1**). None of the examples in the previous paragraph involve extraction, but let us assume that we rephrase the last example to read, ***what is the value of the relation NAME of the products with WEIGHT <10^2 and LENGTH-A < 1?*** Here the response should be **PRODUCT-X**. Now let us look at the process of responding to this request in greater detail.

The set of rallying points, or **accession numbers** as we will call them, that correspond with each of the selection criteria must be identified. The result will be:

WEIGHT < 10^2	:	<FIRST PRODUCT> <THIRD PRODUCT>
LENGTH-A < 1	:	<FIRST PRODUCT>

and the intersection of these sets will result in:

WEIGHT < 10^2 AND LENGTH-A < 1 : <FIRST PRODUCT>

In this case, there is only one value associated with the relation NAME, i.e., PRODUCT-X, (but there could have been a number

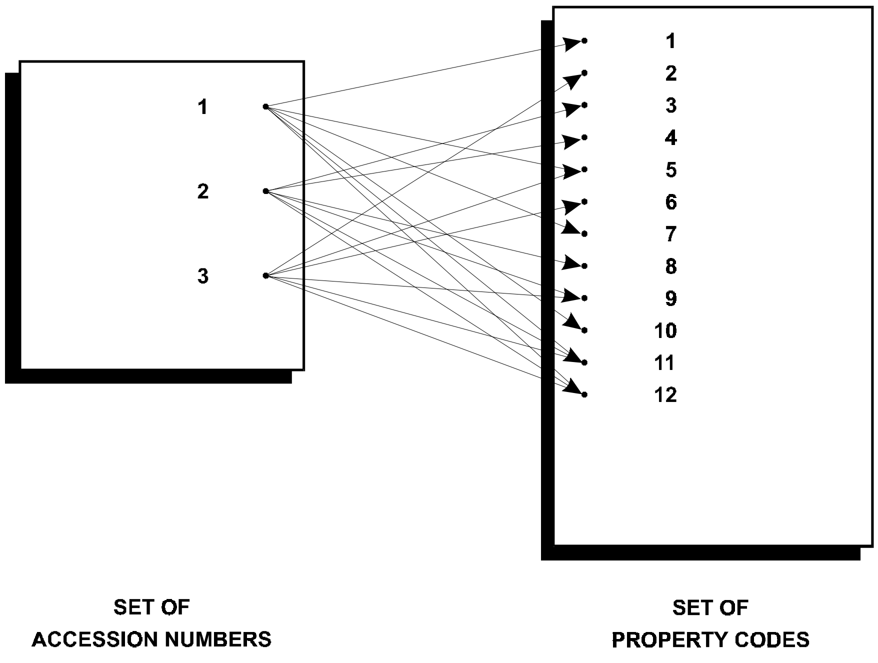

FIGURE 2. THE SAMPLE DATABASE ENCODED

of values). In general, extraction is an answer to the question, ***what-value is associated with the relation X?***

No matter how many occurrences of the property WEIGHT = 13.7×10^2 there are in the database, it will only have to be recorded once. There would be little advantage to this if the representation of the arrows of Figure 1 had a large storage requirement. Because each arrow does nothing more than select one of the properties from the set of properties, a very compact representation is possible. If each of the properties was to be assigned a unique integer, then the binary representation of this integer would suffice to represent its respective property throughout the database. Encoding and decoding would only be required at the user interface. The accession numbers as codes for the rallying points can be treated similarly and the diagram of Figure 1 would become Figure 2.

What makes the proposed IMS feasible is the method chosen to represent the arrows of the two Figures 1 and 2. Two different, but complementary representations are used. One is oriented to the selection function and the other to the extraction function. Let us assume that the accession numbers and the property codes represent the coordinates along two adjacent edges of a rectangular plane. Figure 3 represents this plane, its coordinates and the ●s represent our database.

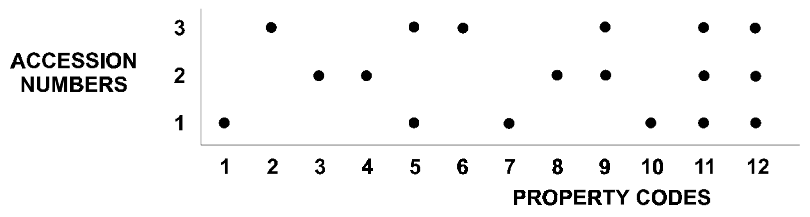

FIGURE 3. THE DATABASE REPRESENTED AS A PLANE

It must be emphasized that Figure 3 illustrates a conceptual database. The actual representation is shown in Figures 4 and 5 and consists of the two complementary representations. Each of these representations is further partitioned into two parts, namely an index or ordered set of descriptors and a set of strings of ordered selection codes. This mechanism makes it possible to move any of the ordered strings to any location within the accessible storage space. The **accession-number-**

index contains descriptors of strings of **property-codes** while the **property-code-index** contains descriptors of strings of **accession numbers**.

The selection function operates entirely on the representation of Figure 5 and the representation of Figure 4 is operated on only by the extraction function.

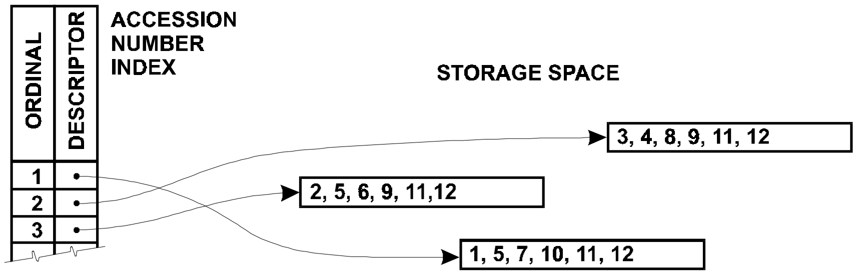

FIGURE 4. ACCESSION NUMBER INDEX MAPPING TO ORDERED STRINGS OF PROPERTY CODES IN STORAGE SPACE

Although it may not be readily apparent, both representations are complete and it is a straightforward process to reconstruct either one from the other. To protect the database from device

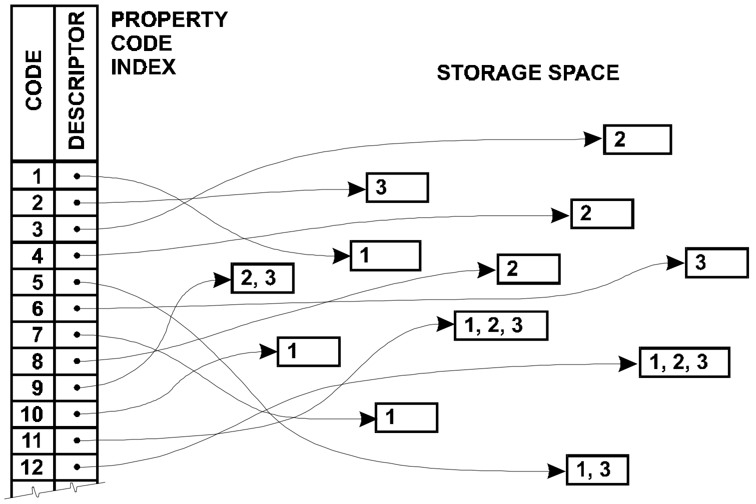

FIGURE 5. PROPERTY CODE MAPPING TO ORDERED STRINGS OF ACCESSION NUMBERS IN STORAGE SPACE

failures, it is desirable to physically separate the two representations comprising the database. In this way, recovery can virtually be guaranteed.

The biggest objection to totally inverted systems is the large amount of work involved should a record change its address. It is precisely because inversion is by address that this is a problem. In the IMS proposed here this aspect has been carefully considered. The proposed system inverts by accession-number and the accession-number remains constant for the objects entire life within the system. The accession-number-index with its descriptors provides exactly the one level of indirection needed to eliminate this and many other problems. This index also serves as the locus of control for individual property-code strings. This in turn allows synchronized updating of the same on-line database by numerous simultaneous users. All updates imply a change to the descriptor since overwriting in place is not allowed. This implies the locking and unlocking of individual entries in the accession-number-index. This continual movement of strings is used as an aid to continuous storage management and garbage collection.

There is a very high degree of security inherent in the encoded representation. The user has no way of determining the code for a value or the accession number of an object. Both items are internal to the IMS. With the exception of a small number of set-theoretic operators that operate on accession-number-strings and the synchronizing primitives needed to lock and unlock parts of the system, the control of the entire system lies predominantly in four routines: ***encode***, ***decode***, ***select*** and ***extract***. Over and above this, the set description routines and user-provided criteria are used to validate data and control all users' access to individual data fields.

The encoding and decoding functions will not be discussed in detail, but some of the considerations will be described. In the conceptual model presented above all properties are handled uniformly. The detailed design does not, however, treat them all in the same manner; classification by attribute (i.e., relation name), and type of data, makes some needed and different encodings necessary. For example, encoded representations of numeric data in the same context, (i.e. bearing the same relation to each of a set of objects), should preserve order; it must be possible to compare the codes to determine comparative rank or equality.

Another example is the treatment of dates. All dates have four inversions: day of the month, month of the year, relative year of the time-span specified for the relation and offset in days from the start of the time-span associated with the relation.

All sets of non-numeric strings have two representations, one specifically for the encoding function and another for the decoding function. This not only provides the redundancy needed to ensure recovery in the event of a malfunction, but also allows the two representations to be ordered differently. This is done to improve and simplify the encoding and decoding processes; the encode representation is ordered on some property of the external representation of the individual strings and the decode list takes the form of the same strings ordered by their codes ranked as binary integers.

The actual representation of the strings of property codes is extremely compact. Only recorded properties are included and no restriction is placed on the number of occurrences of values bearing the same relation to a single object. Each user may have a completely different view of the identical data and may even choose to label the relations differently. Each set of objects requires a set description and either the user responsible for the set will provide and control this description of the set or a database administrator will. Whoever it is that is responsible for a set will also control both public and specific users’ access to that set.

Specific relationships between objects in the same or different sets are themselves treated as sets of data. For example, individual products in our original example might constitute a set of objects and particular reports may comprise another. Each report may require one or more products' data. Regardless of how much data is stored concerning each of these objects, their relationship to one another might make up a completely separate data set. This data-set only has to represent the mapping between the two sets of objects. The individual details may be extracted from the data sets representing the objects.

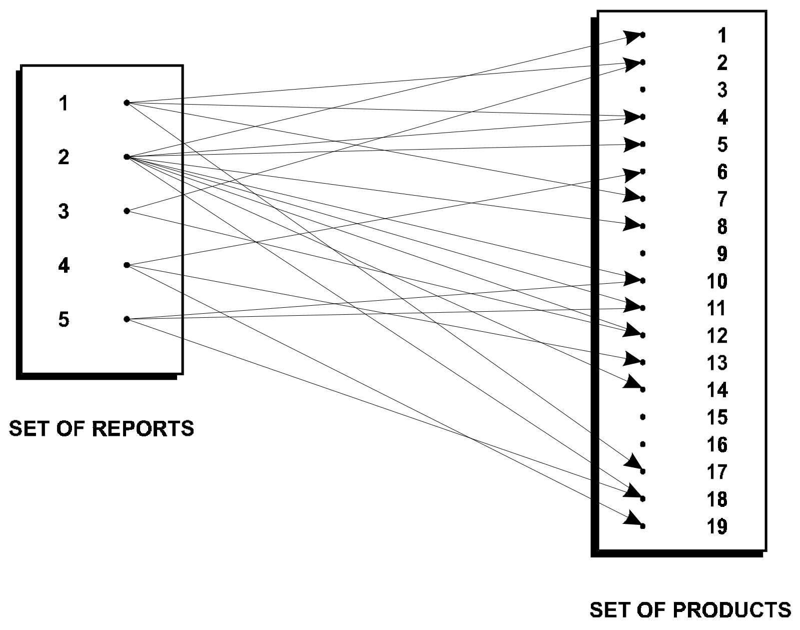

FIGURE 6. A RELATION BETWEEN THE MEMBERS OF TWO SETS

Once again, a dual representation is desirable. Suppose that the two sets of objects in Figure 6 represent our products and

PRODUCT ACCESSION NUMBER

<1, 2>	< 1, 2>
<1, 4>	< 2, 1>
<1, 7>	< 2, 3>
<1,17>	< 4, 1>
<2, 1>	< 4, 2>
<2, 4>	< 5, 2>
<2, 5>	< 6, 4>
<2, 8>	< 7, 1>
<2,10>	< 8, 2>
<2,11>	<10, 2>
<2,12>	<10, 5>
<2,14>	<11, 2>
<2,18>	<11, 5>
<3, 2>	<12, 2>
<3,12>	<12, 3>
<4, 6>	<13, 4>
<4,13>	<14, 2>
<4,19>	<17, 1>
<5,10>	<18, 2>
<5,11>	<18, 5>
<5,18>	<19, 4>

REPORT ACCESSION NUMBER

FIGURE 7. THE RELATION AND ITS INVERSE

reports, and the arrows their relationship. The two lists of Figure 7 would constitute the representation of these relationships.

Depending on the particular characteristics of these relationships, their representation could either be included in the set of reports and the set of products, or it could be recorded as a completely independent set. This is the kind of decision that would be made by either the database administrator or by a system algorithm that monitors usage. Rather more complex relationships that can be derived from those recorded are also possible. The area of familial genealogy provides numerous examples. Consider a data set recording the details of a popula-

<FATHER, SON>
<FATHER, DAUGHTER>
<MOTHER, SON>
<MOTHER, DAUGHTER>

FIGURE 8. THE RELATIONS MAINTAINED FOR A POPULATION

tion. If the relations illustrated in Figure 8 are recorded, then it becomes possible to infer the relationship – (PATERNAL-GRA NDFATHER, GRANDSON), or any other family relationship for that matter.

The birth of a son to any member of the population might make a man a grandfather for the first time. When the new child's data is added to the database including his relationships to his parents, of course, then his relationship to his paternal grandfather would be included in any subsequent requests for data about the relation

(PATERNAL GRANDFATHER, GRANDSON).

The IMS described above is the result of a number of years of careful analysis of user requirements and computer oriented operations. The storage requirements for a comparatively large database can be shown to be extremely modest. The many details are beyond the scope of this brief paper, but the author feels that this IMS concept would be of great value to the user community.

APPENDIX

Sets in the context of the above may be simple or extremely complex.

{A, B, C, D, E}

is a set of elements in which each element has no further complexity. However, the members of a set may be complex sets in their own right, and

{L, M. <N, O, P>, A, Q, {J, K}, B}

illustrates a more complex set. (D. L. Childs represents the level of nesting explicitly in his Extended Set Theory.)

One of the more complex problems in the representation of sets lies in the area of ordering. In a set, in the mathematical sense, no ordering of the members of the set is implied. In the use of sets to represent real world objects, order is in many cases explicitly specified. Order also makes the computer oriented manipulation of sets more simple. Set theory has provided a construction, (due to Kuratowski), for representing ordered n-tuples. Unfortunately this construction is ambiguous when dealing with the ordering of more than two elements. The ordered triple

<X, Y, Z>

may either be interpreted as the ordered pair **<X, Y>** followed by the element **Z** or as the element **X** followed by the ordered pair **<Y, Z>**

Using Kuratowski's construction the former becomes the set

{{{X}, {X, Y}}, {{{X}, {X, Y}}, Z}}

and the latter becomes the set

{{X}, {X, {{Y}, {Y, Z}}}}

To avoid this ambiguity order must be considered as data and must be recorded as such in the representation of ordered tuples. (Childs does explicitly represent order in his Extended Set Theory.)

The sets referred to in the paper may have members which are simple elements, sets or tuples. Order is treated as data and the elements of a tuple that are members of a set are represented by both their value and their order. The value may be a set, a simple element or a tuple.

Every set has a unique name and when one set is a member of another it is represented in the former set only by its name and the fact that it is a set. In this manner the nesting problem is solved.

This paper was originally presented at the ACM SIGMOD Workshop in Ann Arbor, Michigan in 1974.

RELATED BIBLIOGRAPHY

Beitz, E. H., *The-Interpretation of Structured Stored Data Using Delimiters.* Record ACM SICFIDET Workshop, Houston 1970.

Childs, D. L., *Feasibility of a Set-Theoretic Data Structure: A General Structure Based on a Reconstituted Definition of Relation.* Proc. IFIP Congress 1968.

Childs, D. L., *Description of a Set-Theoretic Data Structure.* Proc. FJCC 1968.

Childs, D. L., *Extended Set Theory: A Formalism for the Design, Implementation and Operation of Information Systems.* Set-Theoretic Information Systems Corporation, Ann Arbor, Michigan 1974.

Dewey, G., *Relativ Frequency of English Speech Sounds.* Harvard University Press 1923.

Feldman, J. A. and Rovner, P. D., *An ALGOL-based Associative Language.* CACM Vol-12 No. 8 August 1969.

Halmos, P. R., *Naive Set Theory.* vanNostrand Reinhold 1960.

Kolmogorov, A. N., *Three Approaches to the Quantitive Definition of Information.* International Journal of Computer Mathematics 1968.

Mealy, G. H., *Another Look at Data.* Proc. FJCC 1967.

Preparata, F. P. and Yeh, R. T., *Introduction to Discrete Structures.* Addison-Wesley 1973.

Rovner, P. D. and Feldman, J. A., *The LEAP Language and Data Structure.* Proc. IFIP Congress 1968.

Schwartz, J. T., *Abstract Algorithms and a Set-Theoretic Language for their Expression.* Computer Science Department, Courant Institute of Mathematical Sciences, NYU 1971.

Schwartz, J. T., *On Programming: An Interim Report on the SETL Project-Installment 1: Generalities.* Computer Science Department, Courant Institute of Mathematical Sciences, NYU 1973.

Shannon, C. E., *Prediction & Entropy of Printed English.* Bell System Technical Journal 1951.

Suppes, P., *Axiomatic Set Theory.* vanNostrand Reinhold.

Warren, H. S., *ASL:.A Proposed Variant of SETL.* Computer Science Department, Courant Institute of Mathematical Sciences, NYU 1973.

Zipf, G. K., *Human Behavior & the Principle of Least Effort.* Addison-Wesley 1949.