DATA BASE DESIGN

Polishing Your IMAGE

Presented to:

HP General Systems Users Group
1981 International Meeting
Orlando, Florida
April 27 - May 1

By:

Karl H. Kiefer
Systems Engineer
HP - Englewood, Colorado

I. Introduction: Context for Data Base Design

The motivation for this essay stems from a perceived lack of understanding among professional programmers and analysts, including Image/3000 users, concerning strategies to adopt, and consequences of choices made, designing and implementing data base systems. From a theoretical view, data base technology is intended to overcome inherent limitations and unnecessary costs associated with the use of historically prior file structures and access methods. Namely, the use of indexed files and flat files resulted in systems characterized by physical redundancy of stored data, dependence between programs and data, update and integrity problems, security problems, and inaccessibility to data for unanticipated requirements. Data Base technology promised to overcome these maladies by providing a means by which users would be able to pool their organizations' information into a centralized, independent structure. Applications would be implemented through a common interface to this structure: the data base management system (DBMS). Actual implementation of data base systems in many, if not most, production shops has fallen far short of the promise of the technology. There are two generally related reasons for these developments.

Systems designers have not yet appreciated the proposition that an institution's management of it's information often determines its ability to react to changes in its environment. Biological evolution can, in one important aspect, be understood as a progression from simple to complex forms, and the complexity of these forms can be explained by the notion of information processing. More complex organisms are typically characterized by more sophisticated mechanisms involved in the processing of information. The sophistication of these processes is typically implemented via complex brains and sense organs. The survival success of these organisms is, in large part, made possible by an efficient means of sensing environmental changes and acting accordingly; of processing information. Similarly, the evolution of social organizations, business enterprises and governmental institutions can be understood in terms of these organizations' ability to process information. Simply stated, businesses which fail to manage information efficiently and wisely will,

at best, be less profitable or, at worst, become extinct. Governmental organizations will be needlessly wasteful and, perhaps, fail to provide the service which justifies their reasons for being. This is necessarily so because they lack the capability to act readily according to pertinent changes in their respective environments.

An appreciation of the potential of data base technology to help provide this capability is a necessary, but not sufficient, cause for the success of the technology in realizing its promise. The second reason for its perceived failure is the costly lack of information and guidance from academia and, especially, vendors, with respect to criteria for good data base design and factual information with which designers might more ably evaluate consequences of their choices.

As a result of this lack of appreciation and/or information, implementors of data base systems have historically viewed their DBMS as just another file structure and access method. Typically, an implementor is charged with the responsiblity of getting an application up and running and, perhaps, a choice is made for IMAGE over, say, KSAM according to some vague notion of "fitness" or performance, but from that point on, the DBMS is just another tool in the application implementation.

With respect to IMAGE/3000, some information regarding design and programming choices affecting systems throughout performance has begun to be disseminated to most users. Very little information however, exists elucidating either the criteria for design strategies or the impacts of design decisions when made. This essay, then, is an attempt to provide an outline of design considerations without claiming to know or state all of the costs or impacts of IMAGE/3000 design decisions. Indeed, it is hoped that if users know first what questions to ask, more complete answers may be obtained from actual implementation experiences which, through forums such as this, can supply valuable, shared information. Moreover, it is hoped that the impact of this essay will be applicable not just to IMAGE, but to data base design generically. This, it seems, is particularly desirable since, as we all know, full comprehension of any single product or aspect of our profession is almost certainly an indication of its obsolescence. By the time any of us knows all that can be usefully known about IMAGE data bases, we will surely be rewarded with a completely new DBMS about which we will know next to

nothing, and we can start all over again.

II. Data Design and Relationships

Recent literature on desirable analysis techniques (read: structured analysis) invariably teaches that the first step is a global statement of the functions or objectives of the system. The same holds true for data base design. No technical methodology or checklist of analysis considerations can compensate for less than a thorough conceptual understanding of the functions which are to be implemented using the projected data base. Once these functions are stated and understood, the designer can proceed with the initial phases of design. These consist of the identification of the data items required to process the functions, and an analysis of the relationships which adhere between them.

Systematically associating data items with their functions can be accomplished using a function matrix (see the Data Base Design Kit for the HP 9845C, Hewlett-Packard part #09845-91057) and is a relatively straight-forward task.

Next, it is useful to characterize each item according to two important attributes. An item's VOLATILITY depends on the relative rate at which its contents change value. A part number in an inventory system is not volatile while its on-hand-quantity might be highly volatile. This characterization will impact decisions on performance, integrity of data and storage consideration. An items STRUCTURAL STABILITY refers to the physical way in which it is stored in the data base; i.e., its data type, and length. Ignoring or understating the possibility of structural change can have costly consequences.

The next design decision is fundamental to the success of the data base system. Grouping of items into entries or records obtains a definite relationship between the items which physically binds them together for storage and transfer. Ideally, IMAGE entries should reflect naturally some component of the function with which the related items are associated. An entry describing a part in an inventory system, for example, might naturally relate the following items: part no., description, bin no., quantity on hand, and quantity on order If we define well-organized data as easily accessed, storage space efficient, and easily restructured, then further evaluations are required when grouping items into entries.

RELATION MATRIX

Data Item	Item Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	1																				
	2																				
	3																				
	4																				
	5																				
	6																				
	7																				
	8																				
	9																				
	10																				
	11																				
	12																				
	13																				
	14																				
	15																				
	16																				
	17																				
	18																				
	19																				
	20																				

FUNCTION MATRIX

Function Number:		1	2	3	4	5	6	7	8
Function Description:									
Data Item	Item Number	DATA ITEM FUNCTIONS							
	1								
	2								
	3								
	4								
	5								
	6								
	7								
	8								
	9								
	10								
	11								
	12								
	13								
	14								
	15								
	16								
	17								
	18								
	19								
	20								

figure 2

The following grouping criteria are not necessarily harmonious; that is, increasing the priority of one may decrease the priority of others. Evaluating the grouping choices made according to these criteria will, however, minimize the possibility of costly surprises later on.

1. Group naturally. As indicated above, a natural grouping will simplify the implementation of the associated function.

2. Minimize redundancy. Saves space but may result in more difficult access, complex programming, and lower performance.

3. Group according to volatility. Usually a boon to performance.

4. Group according to entry size. Large entries may require less system I/O but more memory for buffers. Larger entries may obscure the functions to be performed with them. Large entries may obtain more on-line contention for shared data bases and, hence, lower performance.

5. Group for variable iterations. A table or other iterated values such as transactions are perfectly suited to chains. A table might be grouped into an entry; the trade-off is disc storage and memory space requirements versus I/O's.

6. Group according to structural stability. If items which have a good chance of changing structurally are segregated, the impact of change tends to be less severe.

7. Group for security. Security checks by IMAGE at the set level are less costly for performance than item-level checks.

8. Group for multiple views of same data: this may add to redundancy but is usually consistent with natural grouping.

At this point in the design phase, the designer has a preliminary scheme with entries defined and manual and detail sets related as to functional requirements. He may also have discovered that, since IMAGE obtains a network structure, he may need to implement a three or more level hierarchy through implicit programmatic relationships. For example, suppose that a typical manufacturing

application needs to keep track of a final product's subassemblies, which could themselves have subassemblies and so on for several hierarchial levels. This can quite readily be represented in IMAGE through a recursive structure implemented programmatically. The master set contains entries for each assembly. The unique assembly number contains each related subassembly in a single detail set. The detail entry contains, besides the search item, only one other item, namely, the assembly number for that subassembly the entry for which is to be found back in the same parent master! (See figure 3)

Implementing multilevel hierarchical structures not recursive in nature consists of redundantly adding (typically) an automatic master as the intermediate link. In a retail accounting system, for example, several stores can be represented in a master chaining to each department for that store. The concatenation of the store and department numbers then obtain an implicit, symbolic pointer to an intermediate, automatic master which holds the chain heads for individual item entries for that particular department in that particular store. (See figure 4)

IMAGE is no different from any other DBMS in the sense that it is limited in the variety of structural relationships which it can faithfully represent through its own internal pointer mechanisms. And since we can, if we are clever enough, represent virtually any desired relationship if we implicitly design and implement such relationships in our application programs, the designer must ask and evaluate the response to the question, what are the trade offs associated with implicit relationships?

It is useful to formally distinguish between explicit relationships and implicit relationships, the former being those available through the DBMS while the latter are those maintained solely by application programs. Further, it is useful to distinguish between implicit relationships which use symbolic pointers (as in both figures 3 and 4) and implicit relationships which make use of direct pointers. The status array is used by IMAGE to communicate with the user, data descriptive of, among other things, structural information. Through this mechanism, the user can not only access IMAGE entries directly, but also use this data in implementing his own implicit relationships.

The trade-offs associated with IMAGE supported, explicit relationships

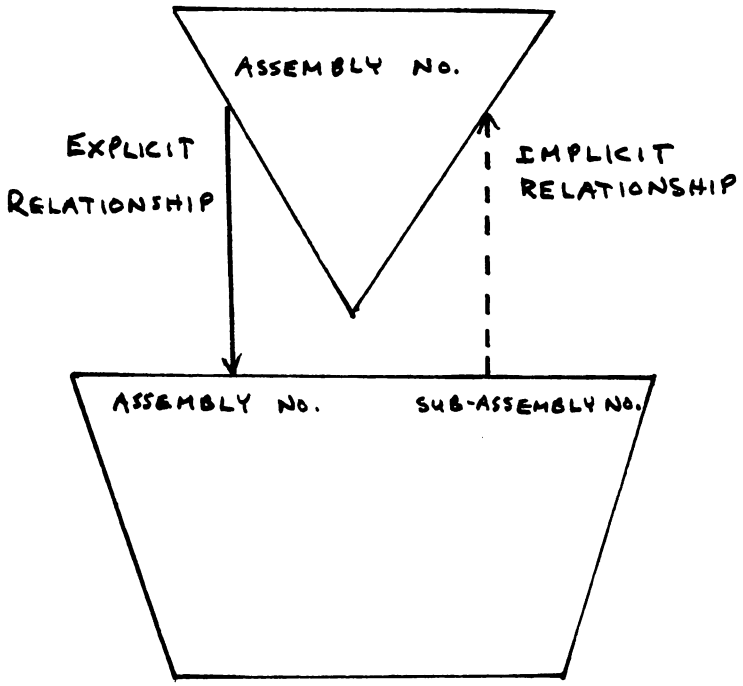

figure 3

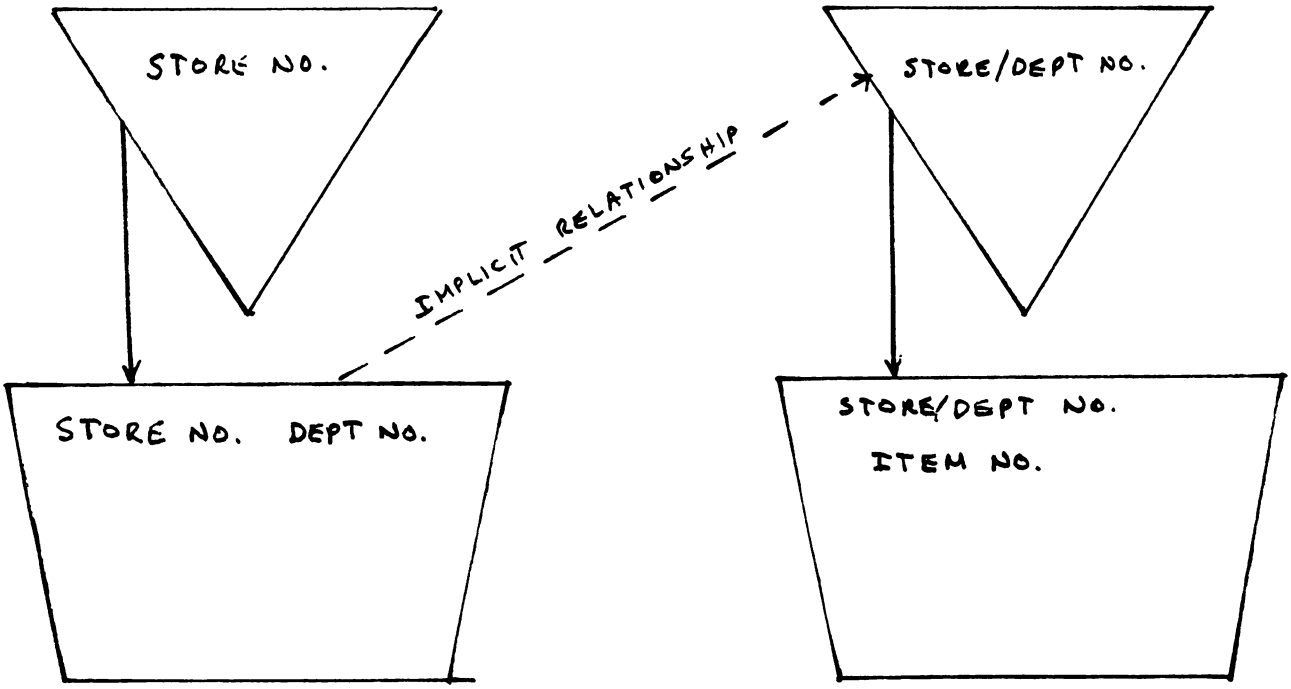

figure 4

include:

1. Limited design flexibility. Faithfully representing all of the organizations' functional relationships may be difficult, if not impossible.

2. IMAGE overhead. Any software tool designed to be general in scope and function has to be intelligent to provide that generality. This translates into overhead and may impact performance (E.G. sorted chains).

3. Low knowledge requirements. IMAGE users are not required to have in-depth structural knowledge. Knowledge is costly.

4. Support utilities. Maintenance of IMAGE data bases is provided by utilities which act consistently with internal structures.

5. Protection from changes. If IMAGE is modified, the DBMS calls are modified accordingly.

The trade-offs associated with implicit relationships include:

1. Unlimited flexibility in representing relationships.

2. Performance may be optimized (Or minimized!)

3. All affected users need to know the structure. High level of knowledge may be required.

4. Structural change to IMAGE may cause unexpected problems.

5. Modifications to and maintenance of user programs tends to be more complex.

6. User-written utilities may be required.

These, then, are some of the general considerations entailed in the first phase of design: identifying the items and their relationships. The second phase

of design investigates the trade-offs associated with optimization for specific characteristics.

III. Data Base Design for Optimization

Many IMAGE data bases are intended to be central to on-line applications for which performance, specifically, response time to human interaction, becomes a primary concern. Since particular IMAGE performance considerations have been discussed elsewhere, we will not attempt to do more than relate general performance issues here. By doing so, we do not wish to minimize the priority of performance as a criterion in data base design. Indeed, that is not our choice to make. We do, however, wish to emphasize other areas of optimization, which, if neglected, can be even more damaging to the success of a data base system.

The pressure of production in the real world obtain, by default if not consciously, the decision to design a data base in order to optimize application development time. The benefits derived from such optimization are, at best, a product to show the end user in a relatively short amount of time. This benefit is almost invariably short term. If design for rapid development is obtained at the cost of other optimizing criteria, it is not long before catastrophes, constant reprogramming, and general dissatisfaction ensue. This is not recommended since effective design does not necessarily preclude quick development. Indeed, the opposite may be a truer consequence.

The relative inexpense of hardware has lessened the demands for optimization of storage space. It is typically cheaper to buy another disc drive than to re-design, or require herculean programming efforts in the interests of mass storage. The disadvantages of optimizing for space usually entail a minimization of redundancy. Even though this is a generally recognized goal of data bases, the realistic application gains increased performance due to more flexible access in the form of, say, redundant search keys in automatic masters. Decreased redundancy also typically entails larger data entries which imply grouping of unrelated items and grouping of volatile with static items. This generally results in larger impacts on application programs if structural changes are required. One benefit which accrues to smaller data bases is by no means negligible, however: the loading and unloading of data bases to magnetic tape for archival or for recovery is a time-consuming task which

is directly related to the size and, in the case of structural reloading (DBLOAD), the organizational complexity of the data base.

Designing a data base with a view towards optimizing performance can be stated as one general rule: Reduce the total amount of work required by the system to process along the primary paths. For IMAGE/3000 data bases this translates typically into minimizing I/O's required to process along the critical paths. Complying with this rule requires, first, that the designer identify the critical paths. This is accomplished by understanding the flows of the applications contending for data base and system resources concurrently. Minimizing I/O activity becomes, first a matter of deciding who can, in fact contend. Can an application be batched if it contends with necessary on-line activity? Setting programming standards may obtain performance gains: locking strategies and standards; item-list processing; inefficient or unnecessary DBMS calls; use of internal record numbers for directed access; use of implicit relationships. Image or HP3000 peculiarities can impact performance: security settings, synonym chains; sorted paths; multiple paths into volatile data set disc drive placement; primary path contiguity on disc; the number of buffers; the sizes of buffers. The costs of performance consists primarily in: knowledge level; redundancy of data and, hence increased storage space requirement; complexity of programs; decreased flexibility in structure resulting in costlier impacts if changes are made; implicit relationships; possible data integrity and update problems due to redundancy and batching of applications not essential to on-line activity.

An appreciation of flexibility as a data base design criterion is, unfortunately, almost non-existent. It is unfortunate because the penalties meted out for failures in this aspect of design are rarely anticipated and often expensive. While performance of data base systems is a highly visible attribute, an inflexible data base structure typically displays its weaknesses suddenly and dramatically. It is usually triggered by an external change in the environment, perhaps as simple as an account number format (zip codes?) or as subtly complex as a slight modification to a standard corporate procedure. The solution may entail a simple organizational unload, modification of the schema, and reload to accomodate the structural change, or it may require many man-days and man-nights of reprogramming, or it may even force an admission that the change cannot be implemented without a total redesign.

A data base is said to be flexible if it is characterized by elastic data structures and elastic data relationships as opposed to inelastic structures and relationships. Elasticity is measured in terms of the ability to withstand change with a minimum of impact.

Redesign for flexiblity, for elasticity, can have significant effects even in trivial cases. Suppose, as is common, that an IMAGE data set is modified by adding a new item to the end of the entry. This is perhaps the simplest of changes to implement. The item is to be used only by a new application so its effects should be minimized. The data base is unloaded. modified, and reloaded. That's it! But wait! Suppose twenty or thirty or forty other programs access that set and, as is likely, for no more reason than programming ease, each program coded each call to DBPUT and DBGET with an item list of "@". Send out for beer and pizza; it will be a late night at the terminal! chances are fairly good, as well, that one or two bugs will creep into the system as a result.

Attaining flexibility in data base design is dependent on an understanding of data bonding and its implications. Bonding of data refers to the relating of data base components through either explicit or implicit relationships. Image data base components can be bonded as follows:

1. Items can be bound by grouping into entries.

2. Sets can be bound by paths.

3. Implicit relationships can form virtually any number of bonds, including those between distinct data bases.

Bonding can be described as tight or loose generally in the order indicated. The greater the number of items bound into an entry, the higher the probability that entry will be impacted by external, environmental change. The designer's first step in incorporating flexibility is minimizing the number of items in an entry. This choice is generally consistent with the functional definition of an entry which obtains an abstraction of some particular object of organizational relevance which an occurence of the abstraction describes, such as a bank account, a product, an oil well, or a transaction. Each item in the entry typically has a specific relationship to all other items in the entry, or, more commonly, to a key item in the entry. If an environmental change impacts the entry, all items in the entry are necessarily impacted naturally,

and programs which process the entry will tend to be impacted naturally by the change in function. If unrelated items are bound into the same entry, environmental changes will unnecessarily impact these items as well as the programs which process perhaps totally unrelated functions.

Data is by definition inelastic in direct proportion to its redundancy. Every occurence of the items is impacted by relevant environmental change. If an item is both redundant and grouped with unrelated items, the impact of change multiplies even more. If a data functionally belongs in more than one entry, the designer needs to consider the reasonableness of combining the entries. Designing for flexibility commands that an item appear in only one data set and that items which serve to implement logically different functions should reside in logically different entries. (It should be clear that while key items are a necessary exception to these rules, the exceptions should be kept to a minimum).

For IMAGE this means that, if flexiblity is the design goal, a master/detail relationship in which the detail contains unrelated data items ought to be resolved into a master related to two or more details. Stated differently, each search item in the respective details should be an abstraction of a different functional object.

In practice these guidelines are not always so straight forward. Suppose, as in figure 5, a data base is used to maintain cost analyses by product. A master is related to three details, each containing cost figures for materials, labor, and transportation, respectively. Since each detail contains items functionally related to historical costs, a designer might reasonably combine the cost items into a separate detail set if he knows that these items are subject to frequent structural change these cost items are uniform and remain uniform through changes. The cost of doing so is an additional path and some implicit structures relating the material, labor, and transportation activity in the new set.

Designing flexibility at the data base level for IMAGE necessarily requires implicit, non-maintained relationships. The same objectives still apply, however. Separate data bases are warranted when the items and sets which comprise them are functionally dissimilar. If subsets of a data base are highly complex in terms of relationships and tightly bound, there is incentive

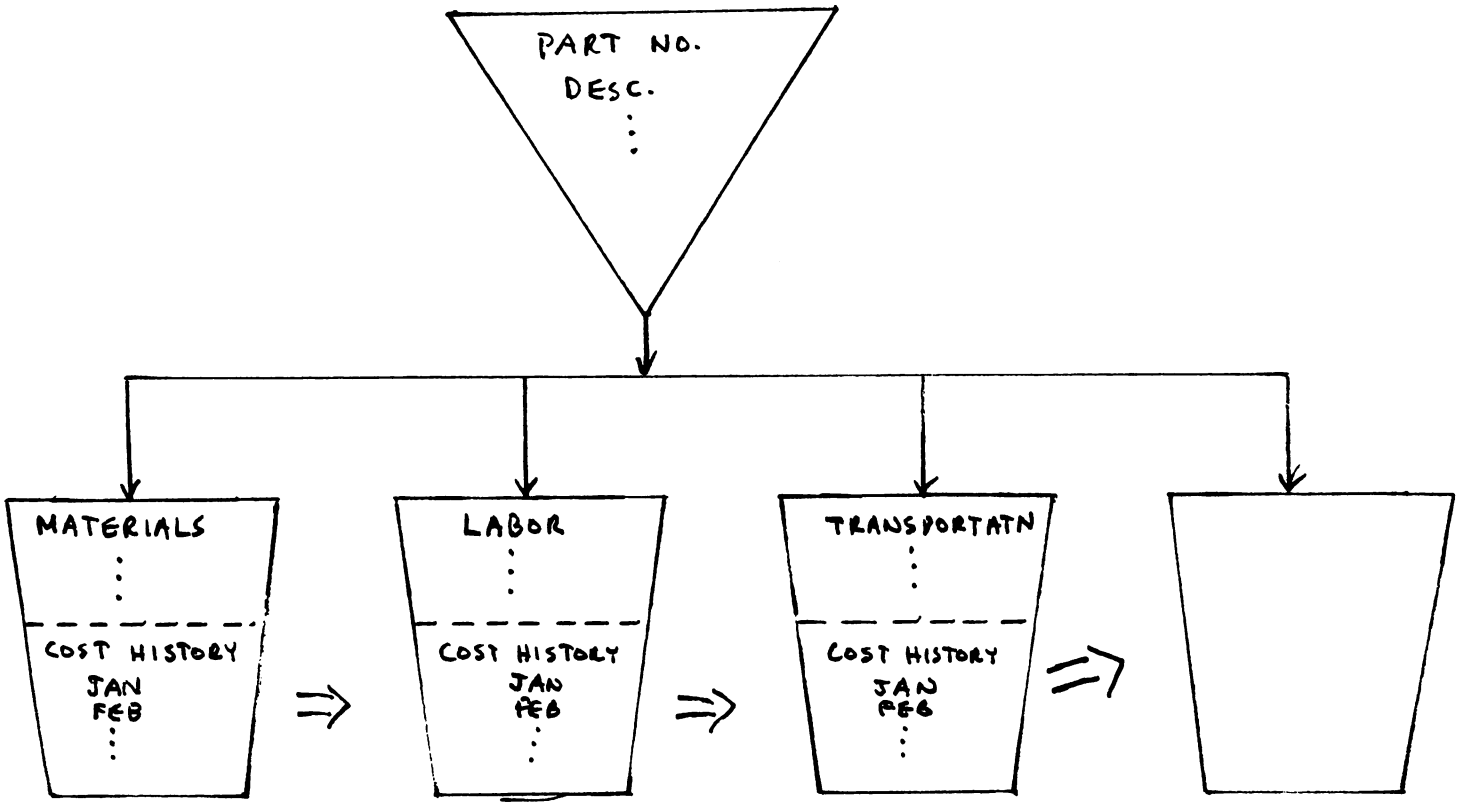
PART NO.
DESC.
MATERIALS
COST HISTORY
JAN
FEB
LABOR
COST HISTORY
JAN
FEB
TRANSPORTATN
COST HISTORY
JAN
FEB

figure 5

to consider breaking up the subsets into multiple, loosely bound data bases in order to minimize the impact of change.

In general, naturally structured data bases tend to be more flexible than data bases structured to easily implement strict user requirements. That is, natural structures tend to faithfully represent processes abstracted from the user's environment and changes to the user's environment will tend to follow naturally, whereas structures created to facilitate particular objectives will tend to be brittle and of narrow scope. The costs for flexibility are not always compatible with performance requirements or desires for ease of development and the proper balance must always be in the designer's mind.

IV. Design Review

As in any thoughtful systems work, design and analysis ought to be an interactive process. In data base design, periodic review with end users, development personnel, and management is the best method for reaching the goal of surprise-free implementation.

The reviews ought constantly to reaffirm the design priorities by evaluating both the reasonableness of the exceptions and, as clearly as can be judged, the costs in terms of other design criteria. End users should be encouraged to distinguish what they want from what they need and to understand, again, the costs associated with sundry features of the system. Analysts and programmers need to understand the requirements for standards in implementation as well as the relative weight of choices to be made during coding and testing. A detailed analysis may require a measure or count of system resource usage demanded by contending processes. Management must be persuaded to consult with DP when considering the adviseability of changes which impact the system. After implementation, periodic monitoring of system usage, performance, and standards enforcement is essential to securing on-going success.

We hope these remarks prove useful both in encouraging discussion of these matters and in proving or disproving their utility in practice. These matters would be far easier to engage if we lived in a world of perfect information with which to make informed, intelligent analysis. The fact is that we do not have anything near to such perfect information and so our tasks are characterized by artistry as much as by technical competence. We will always be

artists to some extent, and so we need to appreciate that the success of the masters depended on skills and knowledge of tools as well as creativity.

References

1. Hewlett-Packard Co., Data Base Design Kit for the 9845B, C, 1980 part no. 09845-91057.

2. Callinane Corp., IDMS Data Base Design and Definition Guide, 1979.

3. Orland J. Larson, IMAGE Data Base Design and Performance Measurement, Abstracts and Proceedings of the HPGSUG, 1978.

4. Alfredo Rego, Design and Maintenance Criteria for IMAGE/3000. Journal of the HPGSUG, Vol III, No. 4, 1980.

5. Berni Reiter, Performance Optimization for IMAGE, Abstracts and Proceedings of the HPGSUG, 1980.