Data base design using IMS/360

by R. M. CURTICE

Corporate-Tech Planning Inc.
Waltham, Massachusetts

TOWARD A DATA BASE DESIGN METHODOLOGY

Data base or file design is the process of specifying how the data is to be located on and retrieved from the various storage media, and what relationships exist among the keys, data elements, records and files of the data base. Occasionally it is useful to distinguish between file design and file engineering. File design is concerned with the logical relationships among file elements, while file engineering deals with physical concerns such as block size, arm movement optimization, and other hardware dependent factors. Although file design necessarily precedes file engineering, very often several iterations between these two are necessary because file engineering considerations suggest a rethinking of many of the file design approaches.

People have been designing data bases with measurable success for as long as there have been random access devices. What eludes us however, is a clear formulation of the precise steps by which the design was constructed. No one has been able to describe a coherent methodology for achieving an optimal or even a good design. Much of data base design thus remains an art.

One prerequisite to the development of a data base design methodology is a clear measure of performance of the resulting design. At first glance we are inclined to measure performance merely in terms of say, daily running time, or total number of accesses, for a given volume of transactions. But often we are quite willing to trade several hours of overnight batch running time to speed up an on-line transaction by a few seconds, or to take an extra disk revolution for a write check to insure data integrity. Moreover, how do we specify a desirable balance between running time and storage size? Or account for periodic reorganizations or future flexibility? It may well be that these factors will have to lie outside the measures associated with initial design methodologies, and a limited objective constructed. Such an objective may be expressed as "given a maximum storage capacity, certain reorganization frequencies, a minimum response time on on-line transactions, etc., then what is the optimum data base design?"

In the end, what we are striving for is something like a deterministic model of data base design, in which the parameters of a particular design situation are input, and a full data base design is output—but we appear to lack the formalisms necessary to describe all the elements involved, as the examination of measures above indicates. Some work has been done on a formal description of a data base structure, and to a degree the COBOL Data Division or something similar would suffice. But very little work has been done on a symbology for transactions, and the resulting "transformations" to the data base they are intended to achieve.

Until such a model, or other data base design formalism, is developed we can only generalize upon past experience in order to construct rules or guidelines toward a design methodology. Several papers present rules of thumb for this purpose.[1–6] These and other discussions of data base design methodology usually make no assumptions about the hardware or software used to implement the design. As more users turn to generalized data base management systems, however, the need arises to identify design guidelines which are specific to a particular system. This paper discusses several data base design guidelines based on the use of IBM's Information Management System/360 (IMS/360) data management package.

IMS/360

The IMS/360 software package includes a comprehensive data management system called DL/I (Data Language I). This system is either being used or carefully considered by users in many large IBM installations for data base applications. One important feature of IMS/360 is that it also includes a teleprocessing

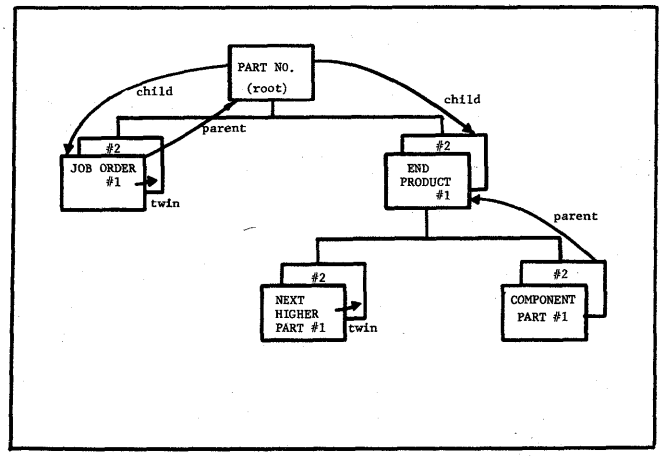

Figure 1—Data base example #1

capability which is intended to facilitate the conversion of initial batch data base applications to an on-line mode. Like most data management systems, IMS enables the user to separate the data base description from the applications programs, thus permitting certain changes to be made to the files without affecting all programs. The system also includes backup and recovery modules as well as file reorganization and statistics collecting utilities. It is clear that for most installations developing large integrated data base applications a generalized data management capability similar in scope to IMS will be required.

In addition, IMS provides a quite general structure with which to describe the data base record (or in IMS terminology, segment) relationships. This structure is a hierarchy of fixed length segments emanating from a root segment for each data base. Figures 1 and 2 both show examples of such structures. Up to 255 segment types, arranged in a maximum of 15 levels, may be specified for an IMS data base. Each segment type may occur any number of times or not at all under a given root. As shown in Figure 1, the immediate subordinate segments are referred to as child segments, the segment they appear under is referred to as the parent, and different occurrences of the same segment type are referred to as twins.

One dominant feature of the IMS data management scheme is that the applications programmer always views the data structure as hierarchical, regardless of the access method employed, or physical location of the data. Thus the four IMS access methods all begin with "Hierarchical"; namely: the Hierarchical Sequential Access Method (HSAM), the Hierarchical Indexed Sequential Access Method (HISAM), the Hierarchical Indexed Direct Access Method (HIDAM), and the Hierarchical Direct Access Method (HDAM). When

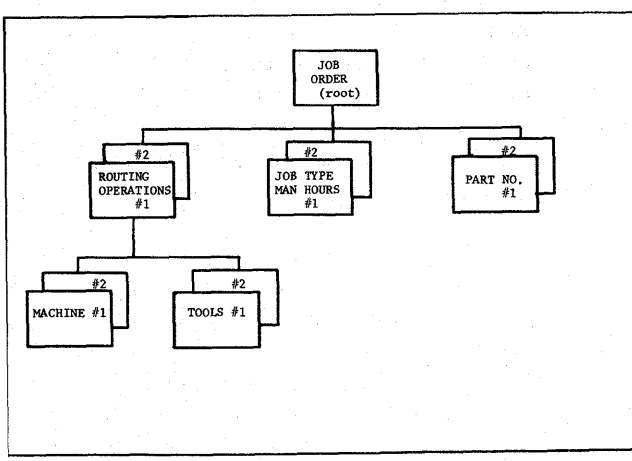

Figure 2—Data base example #2

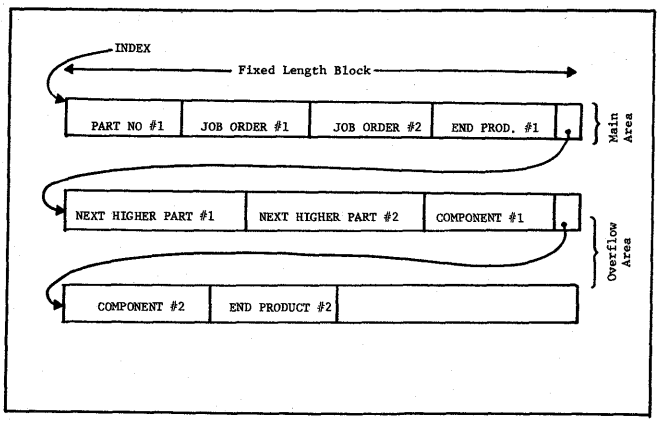

Figure 3—HISAM example

using either HSAM or HISAM, the hierarchy relationships are maintained by physically recording the segments sequentially in a top-down, left to right convention. HISAM is conceptually similar to the indexed sequential access method under O/S. It provides an index to the root segment, and a separate overflow area, as the example in Figure 3 indicates. This figure shows how a sample record from the data base in Figure 1 would be physically stored using HISAM. As many of the segments occurring under a root as can fit into a fixed length block are stored there, while the rest are chained together in other overflow blocks. The segments under a root must be accessed sequentially, and insertion of a new segment causes others to be shifted down. This necessitates periodic reorganization.

HIDAM and HDAM do not record the segments under a root sequentially, but rather allow direct pointers to the children, from the children to the parents

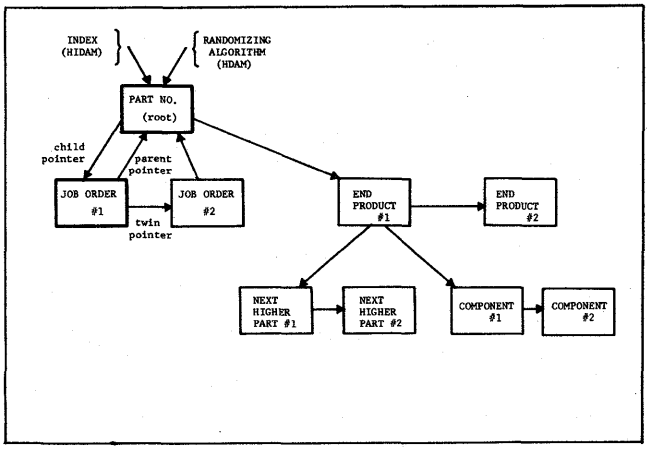

Figure 4—HIDAM and HDAM example

and among the twins under a given root, as shown in Figure 4. In effect, each segment type becomes a file. As the names imply, HIDAM provides an index to the root segments, and HDAM accesses the roots with a user supplied randomizing module.

The ability to specify relationships across data bases is provided by defining a logical data base which is composed of segments from one or more physical data bases. The sample data base assumes there is a requirement to process both parts and job orders as separate entries, but often we need to access one file from the other as well. While this can be done by repeating job orders for each part, and part numbers for each job, redundancy of data results and the programmer must perform a double maintenance task. As an alternative, IMS permits the specifying of a direct access pointer

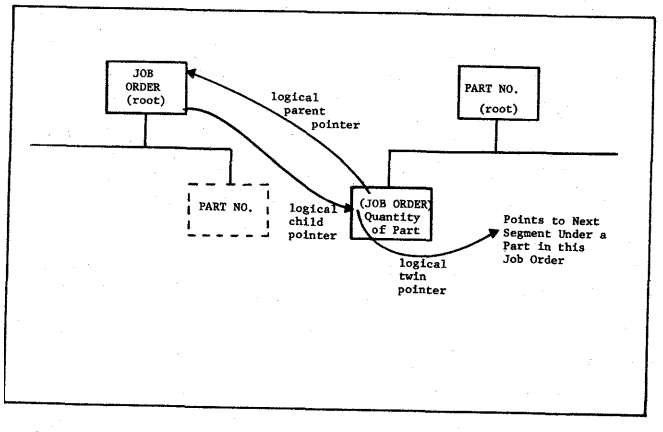

Figure 5—Use of logical pointers

from one segment to another, as shown in Figure 5.* Here, the job order segment in the Part Data Base does not contain the job order number but rather a direct pointer to the root segment for that order number in the Job Order Data Base. Other information, called intersection data, may be recorded in the job order segment in the Part Data Base. While a similar reverse pointer can be made from the part segment under the job, an equivalent capability would be to begin a chain from the job order root, connecting all the segments for this given job order number under the various parts in the Part Data Base. To do this a "virtual" segment (shown in dotted lines) is defined. Once these direct pointers are in place, logical views of the combined data base may be specified. A logical view is a hierarchy of segments which, while not physically related in that hierarchy, can be made to appear so by utilizing the direct pointers. For example, when a programmer accesses the job order segment under a part number,

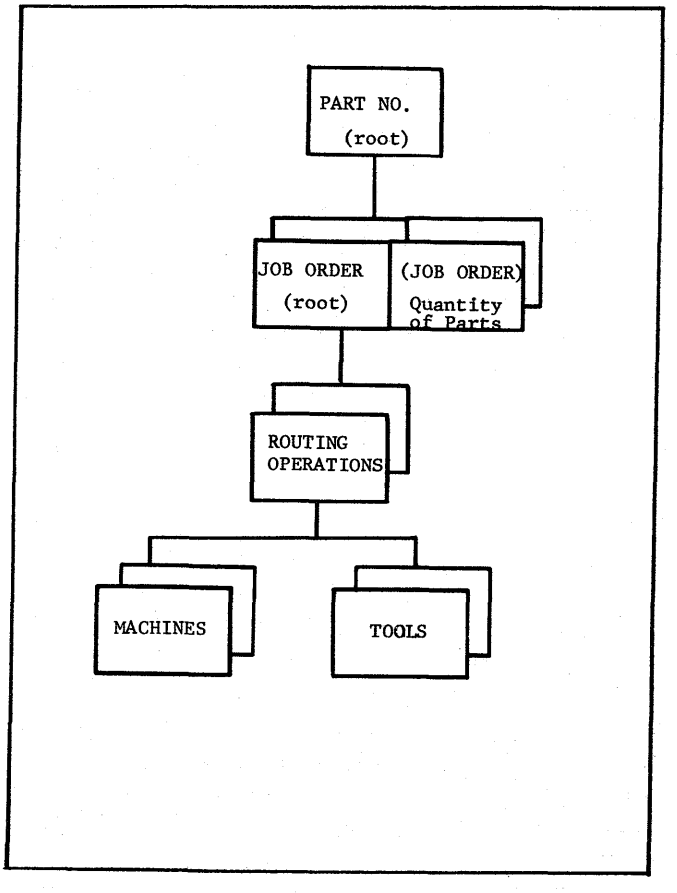

Figure 6—Logical view using logical parent

* These are referred to as logical pointers to distinguish them from the physical pointers used in the primary hierarchy as shown in Figure 4.

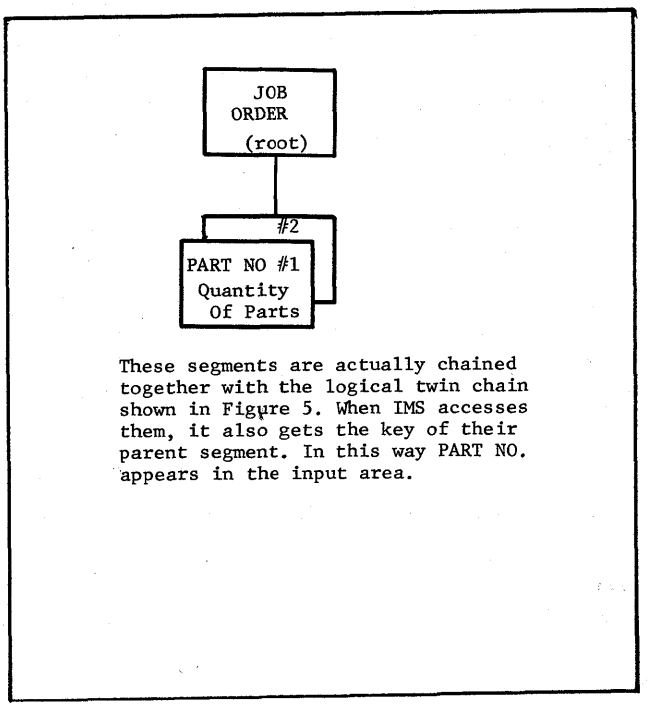

Figure 7—Logical view using logical child and logical twin

the record to appear in the buffer includes the job order root segment in the Job Order Data Base. Another possible logical view is shown in Figure 6.

Using other pointers, a different logical view would appear to the programmer as shown in Figure 7. Here the virtual segment appears to be under the job order root. Since IMS retrieves the key of the parent segment of a logical child segment, the part number appears in this virtual segment as well.

This brief introduction to IMS contains obvious oversimplifications, and the reader is cautioned to refer to the IMS manuals for further detail. Other features of IMS are introduced below as required.

THE DATA BASE DESIGNER

The use of a generalized data base management system, and the desire for integrated data bases, both necessitate centralization of the file design effort, rather than distributing it among the various application programmers, for example. Much has been written about the importance of the data base designer and his role as an interface among the applications teams. Experience with IMS reinforces this view. Moreover, experience indicates that the data base designer must be knowledgeable in the applications at hand. The data base design permeates the applications to such a degree that if any hope of efficiency is to be realized, the data base designer must be able to make positive contributions to program and job stream flow based on optimizing data base performances (relative to the time and/or storage measures as discussed above). Thus, it may be *easiest* from the application programmer's point of view to generate and deal with a particular logical structure, say as shown in Figure 7. But it is the data base designer who knows that each part record under the job number will require at least two physical accesses; he can suggest the duplication of the part number as a trade-off possibility.

Another reason that the data base designer must be familiar with the applications is that application dependent features are actually coded in the data base. For example, IMS permits the optional specification of only unique keys for multiple occurrences of a given segment type. An attempt to add a duplicate key will cause a certain message to be returned and this may be significant in the program logic. Another instance of such dependency concerns the addition and deletion of segments using logical relationships. IMS permits several options with regard to adding or deleting from a logical view. In Figure 7 for example, suppose a program reads a job number, and then deletes a part under this job number. Certain IMS coding may now cause the part number in the Part Data Base to be deleted as well. Obviously, this coding should only be specified after a thorough understanding of the application.

UTILIZATION OF RESOURCES

The price to be paid in the use of a generalized data base management system is some overhead in resource utilization, typically storage space, CPU time, or accesses. It is a mistake to assume this overhead is fixed, and invariant to the data base design. Quite the opposite is true: since each task consumes more resources, the opportunity (and in many cases the necessity) for efficiencies is very prevalent. A methodology of data base design then, depends largely on estimating the "overhead" or cost of certain options, in order that profitable trade-offs can be made. Unfortunately, as with many other systems, the costs associated with various IMS features are not publicized, and in some instances can even be counter-intuitive. In most cases the data base designer must extrapolate from his knowledge about the internal workings of the data management system in order to estimate overhead.

Another input to design trade-off studies is data base statistics. Accurate statistics about the data are vital to an efficient data base design. Note that in some instances, however, the use of a data management system

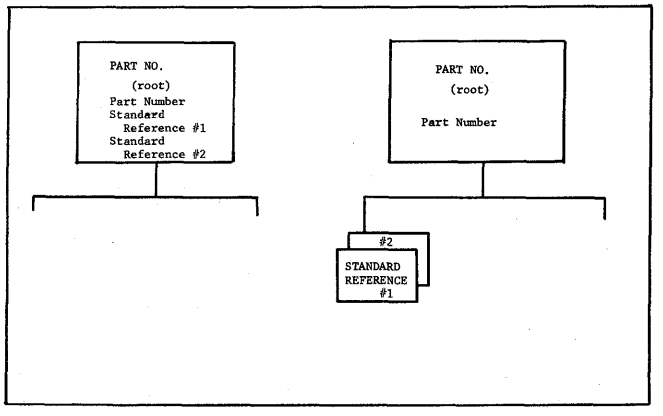

Figure 8—Creating a new segment type

removes the necessity of obtaining accurate statistics prior to data base design. These instances involve precisely the parameters which we are allowed to alter without affecting the applications programs, since these can be changed easily after actual experience with a loaded file has been achieved. The use of IMS permits the data base designer to allocate different physical space to a file, change between HDAM and HIDAM, add a new segment to certain places in the data base, or change blocking factors, all without affecting an application program. But he may not alter fields within a segment, add a new segment to certain places, or modify the logical pointers without incurring some rewriting of the programs. To the degree that the data base is more or less fixed, the trade-offs will only be as good as the accuracy of the statistics upon which they are based.

To illustrate the preceding points, consider this trade-off problem concerning the specification of new segment types. Basically the question is "under what circumstances is it best to create a new segment type?" One case arises when subordinate data may repeat a number of times; it is clear that a new, repeatable, segment is preferable to a large space reserved for the maximum data possible. But what if we know the data can only repeat twice for example? More specifically, suppose we wish to record references to standards for each part, up to a maximum of two standards per part. The trade-off is then between allowing for two such fields within the part root segment, or creating a new segment type subordinate to the part root segment containing a standard reference, as depicted in Figure 8. What factors should be taken into account in making this trade-off? The factors include the following items:

- The IMS storage overhead associated with each new segment occurrence;
- The added complexity of the data base description;
- The unused space if a field is always present but has no value;
- If HIDAM or HDAM are used, the space for the pointer to a subordinate segment from the parent, and the pointer connecting the child twins;
- The accesses necessary to obtain the data in a different segment;
- If HISAM is used, the time to process each new segment type after the block is in core.

Thus the trade-off among storage, accesses, and CPU time must address these factors. To simplify things though, assume we merely wish to minimize storage space, and each standard reference consumes 8 bytes. Each IMS pointer requires 4 bytes, and there is a 4 byte overhead for each segment occurrence. If we allow for 2 standards in the part root we clearly require 16 bytes per part no matter what. But the storage requirements for a new segment type depend on the actual distribution statistics of standards per part. If very few parts have standards then we are better off with a new segment type. Clearly also, if most parts actually do have two standards then one segment is best since each new segment requires at least 12 bytes, 4 for overhead, and 8 for data (if HIDAM or HISAM are used 4 more bytes for twin pointers plus 4 bytes for a pointer from the part root to the new child are required as well). If the actual occurrences lie somewhere in between then more accurate statistics are probably needed for the trade-off to be made. Otherwise, if either method is likely to result in about the same storage requirement, then some other factor, probably number of accesses, would be optimized.

DATA BASE MAINTENANCE

Whereas on-line inquiry or status posting applications are the more interesting ones, it is very often the batch file update and reorganization runs which take up the vast majority of system resources. The file designer must be sensitive to the batch update requirements because these can become system bottlenecks just as easily as the on-line applications. In attempting to optimize the overall system, a very delicate trade-off decision is required.

Again, good statistics make for an informed decision. An especially important use of these statistics is to estimate file sizes and growth. File size estimates are needed since the size of the file will directly affect the time required to backup and restore the file or to reorganize it. This is in addition to the input of file size estimates to hardware configuration planning.

When using a data base management system such as IMS/360, the data base designer must take into account the storage requirement imposed by the system, including pointers, control fields, and indices. Actual experience has shown that storage overhead for pointers and IMS control fields can easily reach 50 percent of the total storage requirement. In applications approaching a billion bytes, this overhead becomes a very costly factor. Not only must the cost for physical storage be borne, but the maintenance load is proportionally increased, resulting in a greater processing requirement. Thus the trade-off between storage and accesses should only be made considering the entire system—batch data base maintenance as well as on-line transaction processing.

A final note then about accesses. Hidden accesses in IMS (i.e., the *average* ratio of logical accesses to physical accesses) has run as high as 4 or 5 to 1. Translating into accesses per transaction, the result often shows that between 20 and 50 physical accesses are required per transaction. More complex transactions such as a Bill of Materials update can require hundreds of accesses per item. One can see here that a data base design which minimizes hidden accesses can affect a reduction in running time of a B/M processor by substantial margins.

CONCLUSIONS

One can argue that a data base management system should not be chosen until the optimal file structure has been identified. In this way a system which supports that structure can be selected—rather than forcing an application into a structure dictated by the system and thereby paying in performance. While this argument has merit, practical considerations often leave no choices open. To some degree this will be the case with many IMS/360 users. It is the only data base management system supported by IBM for large applications. It is one of the few systems now supporting TP. It has many (but not all) of the backup and recovery features needed for large data base applications. Other systems like the Honeywell Integrated Data Store offer file structuring capabilities which may be more suitable to a particular application, but are not implemented on IBM hardware.

While this may sound fatalistic, it points up the need to be especially careful in designing files under these circumstances. Too many users have the view that the use of IMS/360 or any other similar system precludes him from paying much attention to file design—that the system will design the files. The examples above show that this is not the case. The user should view the data base management system as a tool in implementing a design which has been arrived at by taking into account both the applications requirements and the features and limitations of the generalized system. Only in this way can he expect both reasonable performance and overhead.

REFERENCES

1 F H BENNER
On designing generalized file records for management information systems
AFIPS Conference Proceedings Vol 31 1967 Fall Joint Computer Conference

2 N CHAPIN
A comparison of file organization techniques
Proceedings of the ACM 24th National Conference San Francisco California August 1969

3 A M COLLMEYER
File organization techniques
IEEE Computer Group News March/April 1970

4 A M COLLMEYER J E SHEMER
Analysis of retrieval performance for selected file organization techniques
AFIPS Conference Proceedings Vol 37 1970 Fall Joint Computer Conference

5 G G DODD
Elements of data management systems
Computing Surveys Vol 1 No 2 June 1969

6 J K LYON
An introduction to data base design
John Wiley & Sons Inc 1971