An Interactive DSS Tool for Physical Database Design

By: Prashant C. Palvia

Palvia, P. "An Interactive DSS Tool for Physical Database Design," Information Sciences, Vol. 54(3), April, 1991, pp. 239-262.

Abstract:

The design of efficient physical databases is a complex activity involving the consideration of a large number of factors. Because of the complexity, mathematical programming approaches seeking to optimize the physical database have to make many simplifying assumptions; therefore, their applicability is limited. Further, the database designer may want to experiment with design preferences and features not considered by the mathematical optimization approaches. In order to effectively design the physical database, this article describes an interactive DSS tool, which aides the database designer in this task. The database design is accomplished in the context of a high-level abstract model which is capable of being implemented in a variety of DBMSs and file systems. Because of this generic nature of the abstract model, the utility of the DSS tool is enhanced. The interactive tool not only lets the designer experiment with his own designs, but also provides several heuristic optimization procedures to enable the generation of many good designs. The heuristic designs may be used for final physical database design as well as for further experimentation. The paper also includes examples of how the physical design selected using the abstract model and the interactive tool may be implemented on several DBMSs and file systems.

Article:

1. INTRODUCTION

Database design is a challenging and complex activity involving two phases: logical design and physical design. Logical design involves the development of a logical data structure (LDS) for the task domain; and physical design is concerned with developing storage structures for placing data on secondary storage, given a specific LDS. In this paper, we focus on the physical design task and assume that the LDS is given to us based on prior design activity (as in Figure 1, adapted from [5]). Prior works have generally dealt with and developed design/optimization models for specific aspects of physical database design. These works include: index selection [1, 14, 17], file structuring and models of file organization [18, 30], and record segmentation and structuring [15, 17, 23, 24]. Some of these models are reviewed in [29]. Quoting from [19]: "While the attention to individual design problems results in elegant solutions, it is quite plausible that those individual solutions will have to be perturbed when the total database is put together."

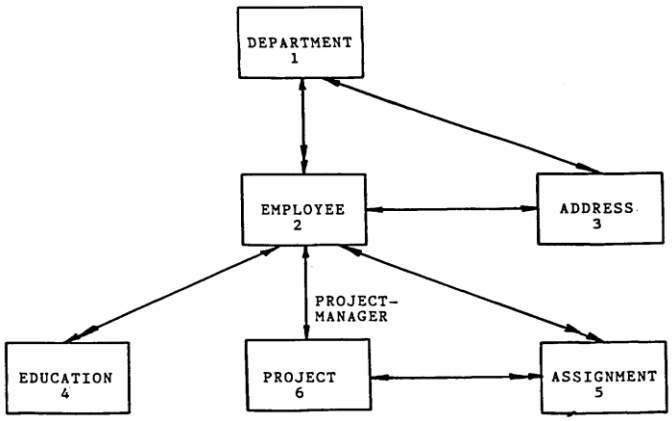

Fig. 1. Example of a logical data structure (LDS).

Researchers have attempted to optimize the total physical database design using mathematical programming approaches, but only with limited success. For example, mathematical programming based optimization is described for hierarchical databases in [28] and network (DBTG/CODASYL) databases in [12, 19, 31]. Optimization algorithms for a general physical database (i.e., one not specifically tied to any particular logical model based system) are de-scribed in [21, 25]. However, all of these algorithms make several simplifying assumptions in order to keep the formulation clean and tractable, thus again perturbing the final optimal solution. To quote from [7]: "Traditional mathematical optimization techniques have not been successfully applied to the global solution of the physical database design problem." As a consequence, several researchers have presented heuristic and interactive procedures for physical database design, e.g., such procedures are described for network systems in [11, 20] and for general physical database design in [6, 7]. Some of these heuristics offer guidelines for design (as in [6]), while others actually try to optimize the design (as in [11, 20]). Note that heuristics cannot guarantee optimality; they only attempt to optimize; however, in this paper, we call it heuristic optimization.

In this paper, we describe an interactive decision support system (DSS) tool for physical database design. The DSS tool, implemented as an integrated software composed of several computer programs, also includes heuristic optimization. The major distinction of our work from prior works is that the model used for physical database design is a generic one, and as such can be used on different types of database management systems (DBMSs). The DSS in conjunction with the model helps the designer make important high level decisions about physical database design, which can be useful for hierarchical, network, relational databases as well as nondatabase file environments (e.g., file-management and COSOL-based systems). The high level decisions include: the number of physical files, their contents, interfile clustering, and general mechanisms for representing relationships.

Our work complements the work on file optimization problems cited in earlier references, i.e., once the high-level decisions are made, then the details of the individual file organizations *can* be decided based on prior models or based on the constraints/options of the implementing DBMS. Of special relevance to our work is the work reported by researchers in [6, 7]. In these works, a DSS methodology is described where file organization problems are first created, using an abstract and generic design model similar to the *one* described in this paper. In subsequent steps of their methodology, specifics of each file organization (e.g., access paths, record segmentation) are decided based on optimization algorithms and heuristics. Their DSS offers extensive help in deciding the details of the file organization, but is rudimentary in terms of deciding which file organizations are to be considered in the first place. Some heuristics are offered in [6]; but these are guidelines based on logical data structure alone, and there is no attempt at optimization in selecting file organizations (except enumeration based on the guidelines). In fact, the authors in [6] state that their guidelines should be overridden depending on characteristics of the retrieval activity. It has been shown in [26] that the characteristics of the retrieval activities are very important in deter-mining the physical database design. The DSS described in this paper includes heuristics which try to optimize the overall high-level design based on all relevant characteristics of the design problem.

The organization of the paper is as follows. In the next section, we describe the generic physical design model used in this paper. After that, we describe the essential features of the computer aided design procedure (i.e., the DSS). This section includes inputs for the software, use of heuristics, and experimentation with the tool. Following that and before concluding the paper, we give examples of how designs produced using the generic design model and the interactive tool may be implemented on network, hierarchical, and file systems.

2. A GENERIC MODEL FOR PHYSICAL DATABASE DESIGN

Generic physical design models for (single) file design have been proposed, in increasing degree of comprehensiveness, in [18], in [30], and in [32], respectively. The file design model of [31] was extended in [3] for physical database design. In addition, there are models and concepts [2, 5, 21, 23, 29] which are, per se, for the entire physical database design or significant parts of it (i.e., without building on file design models). In this paper, we have used an abstract and generic physical design model based on some well-recognized principles

that emerge from the current models. These models suggest two fundamental principles for representing a relationship between two entities. The first principle is well known: Indicate a relationship between two entities by storing appropriate pointers in the entities' instances. The pointers may be in the form of linked lists or inverted lists or some combination. (Note further that the pointers may be direct or symbolic.) The second principle for indicating a relationship is the concept of clustering/aggregation in which all related instances of one entity that are related to an instance of a second entity are clustered with or near the second entity instance. The two concepts yield substantially different physical designs. Our generic model (similar to the model in [5, 6, 7]) captures the spirit of the two concepts and allows for five ways of physically representing two entities *X* and Y and the relationship between them:

a. Create two record types *X* and Y with *X* having pointers to Y.

b. Create two record types *X* and Y with Y having pointers to X.

c. Create two record types *X* and Y with both pointing to each other.

d. Create one record type *X* which will aggregate (cluster) Y instances. Aggregating in the abstract model is actualized by making the related Y instances part of the *X* record.

e. Create one record type Y that will aggregate the *X* instances.

Note that the pointers may be all symbolic or all direct. Also the model allows limited replication if an entity, with an indegree greater than 1, aggregates its related entity. It is worthwhile to note that this model has strong parallels even in commercial DBMSs. For example, hierarchical and network systems incorporate the concepts of pointers and aggregations. Aggregation is supported in IMS by permitting hierarchical segments in the same data set, and in network systems by storing MEMBER record types in OWNER area VIA SET and NEAR OWNER. Relational systems do not allow aggregation at a logical level; however, substantial efficiencies may be achieved by its use at a physical level as reported in [8, 13] In fact many relational systems are now beginning to support clustering (e.g., SQL- and INGRES-based systems and RBASE).

The total number of physical designs, using the abstract model, explode exponentially when the number of entities and relationships in the LDS gets large. For example, with 10 entities in the LDS, there could be over a million design possibilities and with 20 entities, over a trillion design possibilities. Fortunately, we could curtail many of these options at the front end. We found in the many experiments that were conducted with the DSS that although the optimal design is sensitive to the aggregation and pointer options, it is not very sensitive to a specific pointer option. Thus one of the three pointer options can be preselected for each related pair of entities, using judgment, guidelines, or analysis. The analysis is based on a pairwise consideration of related entities. Cost equations were developed for the three pointer options for each related entity pair (cost equations not reported here), and based on that a pointer option was preselected. This pairwise analysis is also a key to one of the heuristics, to be discussed later. Even with one pointer option and two aggregation options for each entity pair, the problem is still large (e.g., a 10-entity LDS may have over 10,000 physical designs and a 20-entity LDS may have over 100 million designs).

With one pointer option and two aggregation options, a physical design can be fully specified by indicating the aggregations alone. A short-form notation can then be used to represent a physical design. In the short-form, only the "aggregator" or "absorber" entity (also called "physical parent") of each entity is named. A root entity does not have a physical parent; so its parent is numbered 0. Table 1 shows some designs for the six-entity LDS of Figure 1.

Note that the first design in Table 1 has all entities stored in their own independent files (of course, with appropriate pointers to indicate relation-ships). This is a design strategy used by many designers. We call it the flat-file design.

TABLE 1

Entities	1	2	3	4	5	6	Explanation
Design 1	0	0	0	0	0	0	Unclustered flat-file design; all entities rooted, i.e., six files
Design 2	0	1	0	2	2	0	1 clusters 2; 2 in turn clusters 4 and 5; entities 1,3,6 are rooted, i.e., three files in physical database
Design 3	0	0	0	0	6	0	Only 6 clusters 5; all entities, except 5, rooted, i.e., five files

3. THE COMPUTER-AIDED DESIGN METHODOLOGY

The design methodology is a series of interactions with the decision support system (DSS) tool. The DSS tool is an integrated software, composed of several computer programs. The four main components of the DSS software are the following:

A. Simulation. The simulation component accepts the problem definition (i.e., the logical data structure, the activities to take place on the database, the computer system characteristics, and a physical database design specification as per the physical design model) and simulates the physical database design as well as the necessary accesses in order to satisfy the activities' requirements. The essential outputs from the simulation are the storage cost required for the physical design and the access cost for satisfying the activities (measured in page accesses from secondary memory). A flow diagram of the simulation component is shown in Figure 2; details are contained in [25].

B. Heuristics. Although the database designer may experiment with different physical designs, a set of heuristics is available in the DSS in pursuit of very good (many times close to optimal) physical designs. Heuristic procedures are known for providing good solutions [10, 33]; such procedures are reported in [15, 17] for specific database problems. Two types of heuristics (actually five heuristics in all) are included in the DSS. The first type of heuristic (called FWI) uses a greedy forward-inclusion procedure for design optimization. It essentially works like this: Select an arbitrary design (e.g., the flat-file design) as the starting incumbent solution. Look at solutions near the incumbent and make the solution with the most improvement as the new incumbent. Reiter-ate the procedure until the incumbent solution can no longer be improved. The second type of heuristic (called PWG) is computationally less intensive and breaks down the complexity by initially doing a pairwise analysis of related entities. The pairwise information is then used again in a greedy manner by the heuristic. Details of the two types of heuristics are available in [25, 27]. Briefly, the FWI heuristic produced the optimal solution in 85% of the test cases, while the PWG heuristic produced the optimal solution in 80% of the cases.

C. Exhaustive enumeration. The DSS tool offers an option of *enumerating* all feasible physical design solutions and prioritizing them by their cost effectiveness. The physical designs are enumerated in a brute force manner and then costed by the simulator. Note that this is a practical option for only relatively small databases (say, with less than 10 entities in the LDS), as with larger databases the enumeration becomes computationally infeasible. (This is precisely the reason for the DSS to have heuristic analysis and "what-if' capabilities.)

D. Pairwise analysis of entities. A pairwise analysis of related entities is conducted primarily for establishing pointer directions among related entity pairs. A secondary purpose of the pairwise analysis is to aid in the initial analysis required in the second type of heuristic (as discussed earlier). In the pairwise analysis, the software examines only the queries focusing on each entity pair and computes relative measures of costs for the three pointer options as well as the two clustering options. These costs become the basis for selecting the proper pointer directions. The selected pointer option cost and the two clustering option costs are used in pruning many of the alternative solution branches in the second type of heuristic. Many cost equations were developed for the pairwise analysis and are reported in [25].

Having described the main components of the software, we now go on to describe in detail the various interactions a designer may have with the DSS. Figure 3 shows an example of the overall interaction with the DSS, for the LDS of Figure 1; details are in subsequent figures.

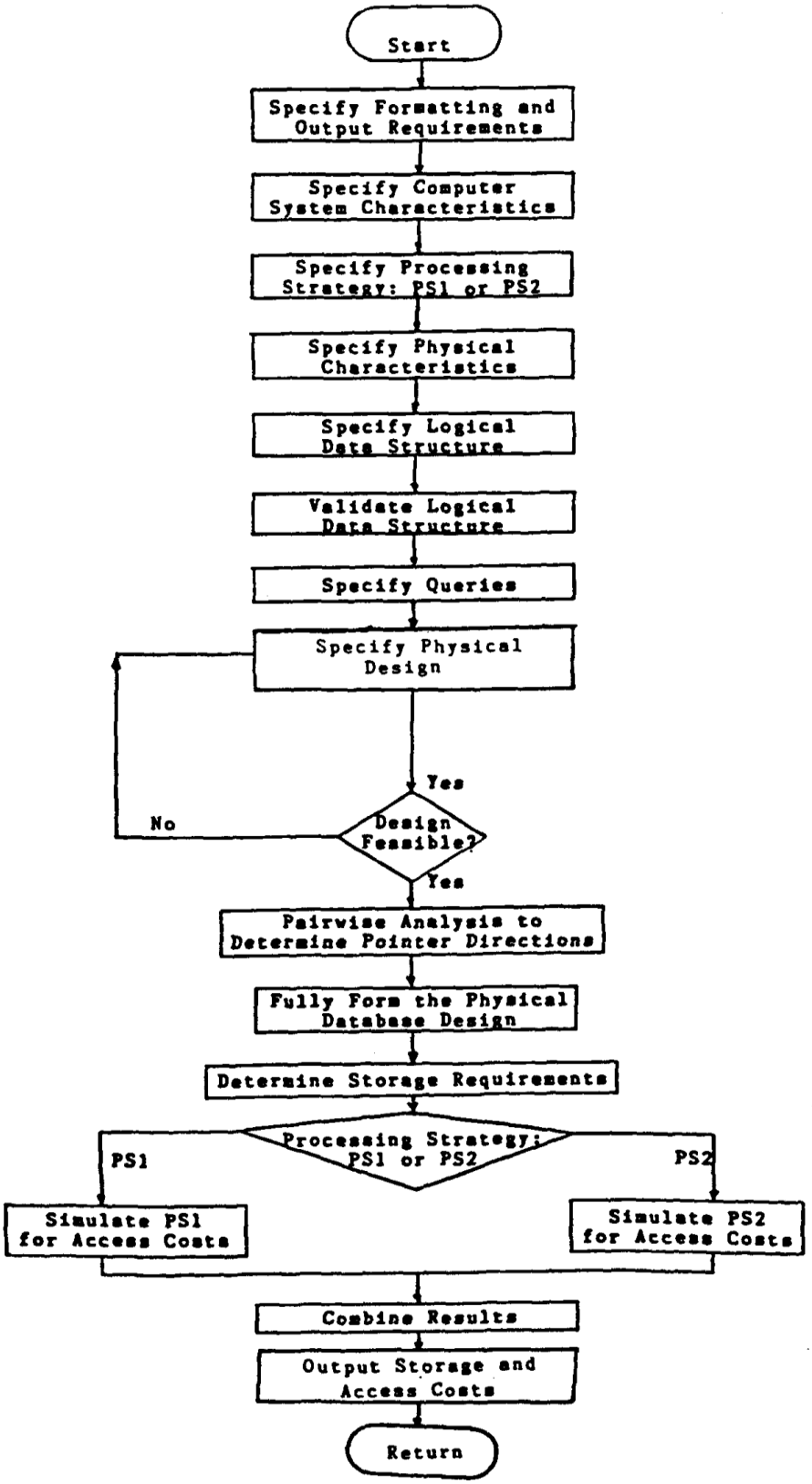

Fig. 2. Main components and line diagram of the simulator.

(A Session)

```
 DO YOU WISH TO USE QUICK MODE FOR ANALYSIS?
?  y
 DO YOU WANT DETAILS FOR EACH FEASIBLE STRUCTURE?
?  n
 ENTER COMPUTER SYSTEM CHARACTERISTICS.
 ---------------------------------------
 ARE COMPUTER CHARACTERISTICS ON FILE?
?  y
 WHAT KIND POINTERS - 1:  SYMBOLIC, 2:  DIRECT
?  1
 DO YOU WANT TWO-WAY POINTERS ONLY?
?  y
 ACCESS PATH - 1:  RAND 2:  SEQ
?  1
 WHAT MODE OF FILE SEARCH(ACCESS)?
 1:  LARGE MEMORY—ONE FILE ONCE, 2:  ONE FILE MULTIPLLY
?  2
 NOW ENTER LOGICAL DATABASE STRUCTURE.
 ---------------------------------------
 IS LOGICAL DATA STRUCTURE ON FILE?
?  y
 ---------------------------------------
 NOW ENTER ACTIVITIES ON THE DATABASE.
 ---------------------------------------
 IS ACTIVITIES DATA ON FILE?
?  y
 ---------------------------------------
 WHICH HEURISTIC YOU WISH TO USE?
 1, 2 OR 3 : ABSORPTION INDEX HEURISTICS
 5 : FORWARD INCLUSION, 6 : FORWARD/BACKWARD
 8 : ALL , 9 : NONE
?  9
 DO YOU WISH TO ENTER YOUR OWN PHYSICAL STRUCTURE?
?  y
 NOW ENTER DATABASE PHYSICAL STRUCTURE
 ---------------------------------------
 DO YOU WISH TO ENTER QUICK FORM INPUT?
?  y
 ENTER ROOT/FATHER FOR EACH ENTITY IN ORDER.
?  0  0  0  0  0
    TOTAL COST =18912.
 DO YOU WISH TO ENTER YOUR OWN PHYSICAL STRUCTURE?
?  n
 DO YOU WISH AUTOMATED ANALYSIS OF FEASIBLE STRUCTURES?
?  n
 DO YOU WISH ANOTHER SET OF ACTIVITIES ANALYZED?
?  n
 DO YOU WANT TO LOOK AT ANOTHER PROBLEM (LDS)?
      STOP
```

Fig. 3. Example of all inputs for the DSS.

3.1. LEVEL OF DETAIL

The user of the system has control over the amount of detail to receive from the interaction. Potentially, the amount of information can be over-whelming; so two levels of detail are provided in the presentation of information (question 1 and question 2 in Figure 3).

3.2. COMPUTER SYSTEM CHARACTERISTICS

Characteristics of the computer system, where the physical database is to reside, are described next. The relevant characteristics are: page size in characters, direct pointer size, storage cost, and access cost (as shown in Figure 4). Note that the objective function can be altered by selecting appropriate values for storage and access costs. The user/designer has the option of specifying these directly to the DSS or storing them in a separate file. If stored in a file (as in the Figure 3 example), then the DSS would retrieve the file.

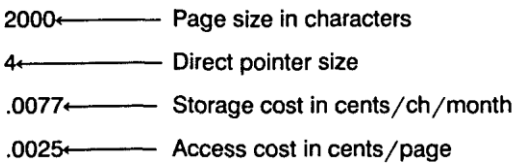

Fig. 4. Computer input for computer system characteristics.

3.3. PHYSICAL CHARACTERISTICS

Certain physical characteristics of the environment are described next. First the user specifies whether the pointers are direct or symbolic. Second, a choice is allowed whether to make all pointers two-way between files, or let the system/user decide the best pointer directions. The system would decide pointer directions based on pairwise analysis of related entities. Next, the user specifies whether the access paths to files are random or sequential. Finally, the user specifies whether the buffers (memory) are extremely large or normal-sized. This last factor affects the query processing strategy. (Briefly, with very large buffers, many searches of a single file can be combined into one search. In this single search, all desired records from the file can be retrieved and stored in the buffer. Details are in [25, 27].) Note that the common selection for the last three factors are nonmandatory two-way pointers, random access path and normal buffer sizes.

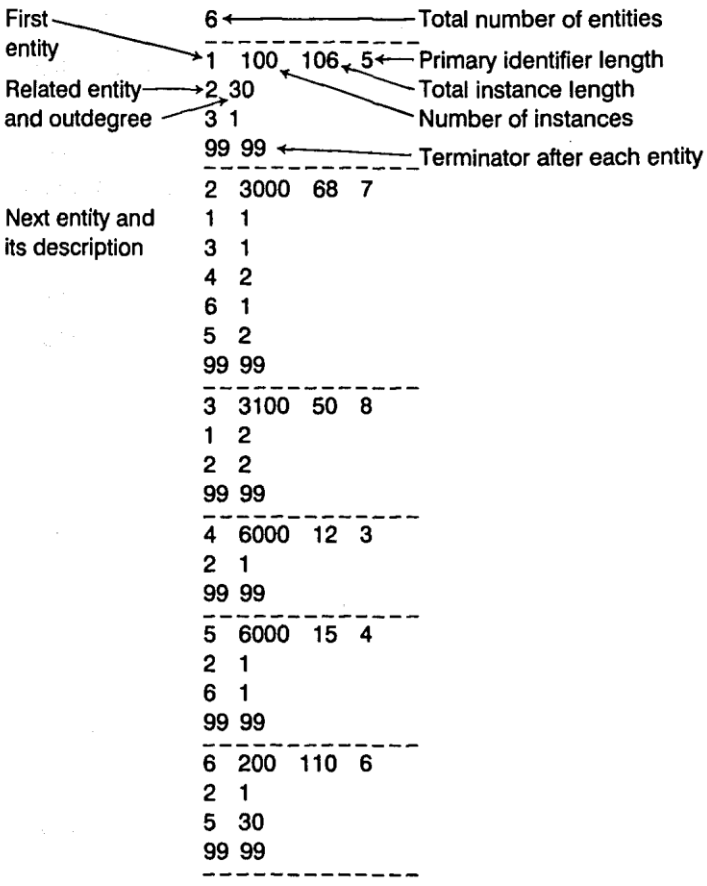

Fig. 5. Computer input for logical data structure.

3.4. LOGICAL DATA STRUCTURE

As before, the logical data structure may be entered directly or via a file. As shown in Figure 5, first the number of entities in the LDS is specified. Then, for each entity, the entity number, the number of entity instances, the length of entity instance, the length of its primary key, the entities with which it is related to, and the outdegree of each relationship are specified.

3.5. LOGICAL DATA ACTIVITIES

The logical data activities are defined next (Figure 6). For each activity, specification is made whether the activity is retrieval (query) or update. Following that, the frequency of the activity and the number of entities addressed by the activity are entered. Then the traversal path of the activity over the entities it addresses is

specified. There is one line input for each entity addressed by the activity. This line contains the current entity in the traversal path, the prior entity in the traversal path, the preselection percent at the current entity and the postselection percent at the current entity. The preselection refers to accessing only selected instances of an entity and postselection refers to selecting among the already accessed instances. It is assumed and expected that the designer would be able to translate any query specification into the analytical form described here.
At this point, the problem specification is complete. Now, the designer will interact with the DSS in pursuit of good physical designs. First, if pointer directions were to be analyzed by the system (see Section 3.3 above), then the pairwise analysis will be invoked, and will do the following.

3.6. ESTABLISHING POINTER DIRECTIONS

As stated earlier, the pairwise analysis component takes each entity pair (which is related by a relationship), looks at the queries focusing only on this pair and computes relative measures of costs for all three pointer options for the pair. The program will either automatically select the least cost option or would display the costs to the designer. The designer can then intelligently select a particular pointer option for each related entity pair. There may be other reasons in selecting a particular pointer option which the designer may want to use (e.g., in relational database design, the pointer is symbolic from the child entity to parent entity). In any case, in our many interactions, we found that a particular pointer option did not make a significant difference in the optimal physical design or total costs. This observation is also documented in [26].

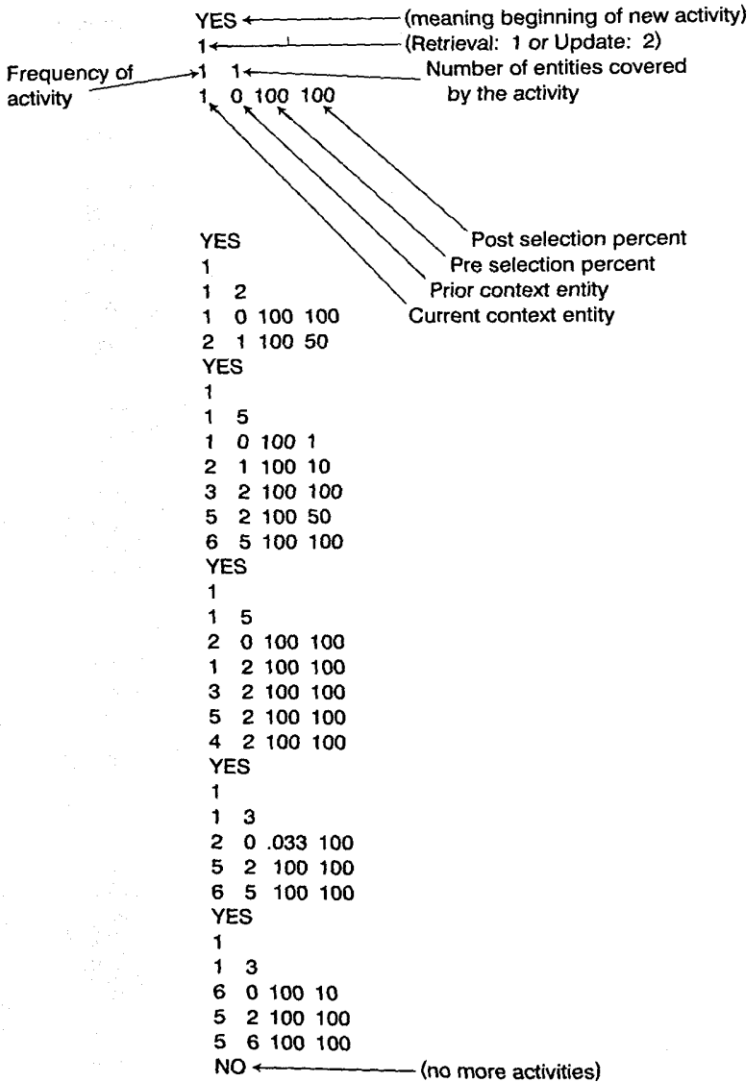

Fig. 6. Computer input for logical level activities.

3.7 HEURISTIC DESIGN SELECTION

At this point, the designer is ready to experiment with the program and evaluate different physical design strategies. One option is to use one or more heuristics in search of good (close to optimal) physical designs. The

designer has five heuristics available (see Figure 3). The last two are based on applying generic principles of heuristic optimization using an iterative greedy approach, where, at each iteration, the design offering the most improvement is selected. In the first three heuristics, a pairwise analysis of related entities is first conducted and that information is judiciously used in developing the heuristics. The third of the first three heuristics augments greedy principles with the pairwise entity information. Based on our experience, we recommend the use of the third heuristic (pairwise greedy, PWG) and the forward inclusion (FVVI) heuristic. As reported in [25, 27], the PWG heuristic, compared to FVVI heuristic, requires less computational effort at the expense of slightly reduced optimality. Briefly, the computational effort of the FWI heuristic was 30-100% greater than that of the PWG heuristic in the test cases. It is recommended to use the FWI heuristic for small to medium problems (say, with less than 30 entities) and the PWG heuristic for very large problems (greater than 30 entities).

```
FILES IN THE PHYSICAL STRUCTURE =3

FILE   1 CHARACTERISTICS:
RECORD SIZE =   3354 NO. OF RECORDS =      100 FILE SZ =    335400
CONTAINS   3 ENTITIES: 1  2  4

FILE   2 CHARACTERISTICS:
RECORD SIZE =    133 NO. OF RECORDS =     6000 FILE SZ =    798000
CONTAINS   2 ENTITIES: 5  6

FILE   3 CHARACTERISTICS:
RECORD SIZE =     66 NO. OF RECORDS =     3100 FILE SZ =    204600
CONTAINS   1 ENTITIES: 3

FOLLOWING FILES REQUIRED FOR ACTIVITY 1
       FILE   1      PAGES =     168 TIMES =       1
    ACTIVITY   1 REQUIRES       168 PAGES.
  PAGES PER MONTH =         168
FOLLOWING FILES REQUIRES FOR ACTIVITY 2
       FILE   1      PAGES =     168 TIMES =       1
    ACTIVITY   2 REQUIRES       168 PAGES.
  PAGES PER MONTH =         168
FOLLOWING FILES REQUIRED FOR ACTIVITY 3
       FILE   1      PAGES =     168 TIMES =       1
       FILE   3      PAGES =       1 TIMES =       3
       FILE   2      PAGES =       2 TIMES =       3
    ACTIVITY   3 REQUIRES       177 PAGES.
  PAGES PER MONTH =         177
FOLLOWING FILES REQUIRED FOR ACTIVITY 4
       FILE   1      PAGES =     168 TIMES =       1
       FILE   3      PAGES =       1 TIMES =    3000
       FILE   2      PAGES =       2 TIMES =    3000
    ACTIVITY   4 REQUIRES      9168 PAGES.
  PAGES PER MONTH =        9168
FOLLOWING FILES REQUIRED FOR ACTIVITY 5
       FILE   1      PAGES =     168 TIMES =  1
       FILE   2      PAGES =       2 TIMES =  1
    ACTIVITY   5 REQUIRES       170 PAGES.
  PAGES PER MONTH =         170
FOLLOWING FILES REQUIRED FOR ACTIVITY 6
       FILE   2      PAGES =     399 TIMES =       1
       FILE   1      PAGES =       2 TIMES =      20
    ACTIVITY   6 REQUIRES       439 PAGES.
  PAGES PER MONTH =         439

************************************************************
      TOTAL STORAGE SPACE IN CHARACTERS =     1338000
      TOTAL PAGES REQD FOR THE ACTIVITIES =     10290
      TOTAL COST OF USING THE STRUCTURE =       10290.
************************************************************
```

Fig. 7. Detailed outputs from the DSS.

The heuristics generate physical designs in short form. Each physical design is then evaluated for storage costs, access costs, and total costs, using the simulator in the program. The simulator determines storage requirements in a straightforward manner. For determining access costs, the simulator first has to determine the files containing the required data for each query and then it simulates the accessing of the files in order to obtain the total number of page accesses. If summary results are desired (by choosing the level of detail, earlier), then the final result will be a total cost for the heuristic design. If more details are asked for, then the details of each

design, the storage requirements, and page accesses will be reported, as in Figure 7. The designer also has the option of not only getting the single best design, but also several good designs listed in rank order.

3.8. EXPERIMENTATION WITH OWN DESIGNS

The designer also has the option of coming up with his own physical designs and let the simulator evaluate them for their costs. These designs may be based on intuition, experience, and/or designs suggested by the many heuristics described earlier. The simulator offers the benefit of conducting sensitivity analysis and pretesting different designs before final implementation. For example, the designer may take one of the designs suggested by a heuristic and then make certain changes to incorporate his own wisdom or local constraints. The effects of these changes can then be readily analyzed. Each design can be specified in short-form, or in long-form (as in Fig. 8). Again, the simulator will evaluate each design, evaluate its feasibility, and present summary or detailed results, as requested. We anticipate that this iterative interface will be very useful for "what-if" type of sensitivity analysis.

(a)

```
 NOW ENTER DATABASE PHYSICAL STRUCTURE
 ------------------------------------
 DO YOU WISH TO ENTER QUICK FORM INPUT?
?  n
 TYPE FILE NUMBER. AT END TYPE 99.
?  1
 TYPE ENTITY NUMBER, ENTITY FATHER. AT END TYPE 99  99.
?  1   0
?  2   1
?  4   2
?  99   99
 TYPE FILE NUMBER. AT END TYPE 99.
?  2
 TYPE ENTITY NUMBER, ENTITY FATHER. AT END TYPE 99  99.
?  5   0
?  6   5
?  99   99
 TYPE FILE NUMBER. AT END TYPE 99.
?  3
 TYPE ENTITY NUMBER, ENTITY FATHER. AT END TYPE 99  99
?  3   0
?  99   99
 TYPE FILE NUMBER. AT END TYPE 99.
?  99
 ------------------------------------
```

(b)

```
 NOW ENTER DATABASE PHYSICAL STRUCTURE
 ------------------------------------
 DO YOU WISH TO ENTER QUICK FORM INPUT?
?  y
 ENTER ROOT/FATHER FOR EACH ENTITY IN ORDER.
?  0  1  0  2  0  5
```

Fig. 8. Computer input for physical design.

```
 DO YOU WISH AUTOMATED ANALYSIS OF FEASIBLE STRUCTURES?
?  y
 DO YOU WANT TO EXAMINE ALL FEASIBLE STRUCTURES?
?  n
 HOW MANY BEST SOLNS (LT 20) YOU WANT?
?  10
 TOTAL FEASIBLE SOLUTIONS =176
                                    1   2   3   4   5   6
                                  ---------------------------
      STRUCTURE   1  :            0   1   0   2   2   0
COST =       4717.00
      STRUCTURE   2  :            2   0   0   2   2   0
COST =       5145.00
      STRUCTURE   3  :            2   3   0   2   2   0
COST =       5604.00
      STRUCTURE   4  :            0   3   0   2   2   0
COST =       5700.00
      STRUCTURE   5  :            0   1   0   2   2   5
COST =       6301.00
      STRUCTURE   6  :            2   0   0   2   2   5
COST =       6564.00
      STRUCTURE   7  :            0   1   0   2   2   2
COST =       6990.00
      STRUCTURE   8  :            0   0   0   2   2   0
COST =       7011.00
      STRUCTURE   9  :            0   3   0   2   2   5
COST =       7058.00
      STRUCTURE  10  :            2   5   0   2   0   0
COST =       7167.00
```

Fig. 9. Top best designs based on exhaustive enumeration.

3.9. AUTOMATED EXHAUSTIVE ENUMERATION

As was described earlier, the DSS also offers an option of enumerating all feasible physical design solutions. If this option is selected, then the software enumerates each physical design in a brute-force manner, determines if it is feasible, and, if feasible, computes the storage and access costs. It then displays the requested number of best designs in rank order (Figure 9). Although this option may be used for a small LDS (say, with less than 10 entities), it is computationally infeasible for larger problems. As stated earlier, this is where the capabilities of the DSS (i.e., heuristic analysis and "what-if' sensitivity analysis) become especially useful.

This completes the description of the iterative tool for the design of physical databases. Once again, this decision support system tool aides the designer in the physical database design process, provides heuristic optimization, allows "what-if' kind of experimentation, and includes exhaustive enumeration option for small-sized problems. We now discuss how the designs and the high-level physical database decisions made using the tool may be incorporated in database and file systems.

4. IMPLEMENTATION

The purpose of implementation is to create the physical schema on commercial DBMSs on the basis of physical designs generated by the interactive tool described above. As stated earlier, the tool is based on an abstract physical design model. Many hierarchical systems, DBTG systems, some relational systems, and many file management systems do include the essential features of the abstract design model; thus the design generated using the iterative tool can be translated for many of these systems. Further, note that the methods of this paper may be used in the creation of the physical database or its reorganization. For reorganization, statistics must be kept on the usage of the data. Periodically, these statistics should be used to generate the heuristic design, and the performance degradation of the existing design from the heuristic design should be computed. The physical database should be reorganized when the cost of performance degradation exceeds the cost of reorganization.

We describe implementation on network (DBTG) based systems, hierarchical systems (specifically IMS), and CONOL-based file systems. These systems generally include physical constructs in their logical models; so implementation can be largely shown by the schemas. Relational models allow only flat-files at a logical level. Clustering is now allowed in certain specific relational systems at the physical level, as was described earlier; however, the description of these specific relational products is outside the scope of this paper.

4.1. IMPLEMENTATION ON A NETWORK SYSTEM

Corresponding to the pointers in the abstract model, there are DBTG implementations with linked list of pointers from parent to children records, and owner pointers in children (as in IDMS), and also proposals for a pointer array in which owner record has an array of pointers to its members. The aggregation affect can be obtained by storing members of a set near the owner by using the location mode of VIA. Note that the DBTG model requires pointers even when the members are stored "near" the owner. Details of DBTG can be found in [4, 9, 22].

The heuristic design for the six-entity (Figure 1) LDS with a specific set of activities was (in short-form): 0 1 0 2 2 O. We proved by exhaustive enumeration that this was also the optimal design. This design implies that the Employee entity be clustered by the Department entity and the Education and Assignment entities be clustered by the Employee entity. Further Department, Address, and Project are "root" entities: They are not clustered by any other entity. Figure 10 shows the DBTG schema required to represent the LDS as well as the key physical design characteristics. Some explanations of the schema follow. There are three separate areas corresponding to the root entities Department, Address, and Project. The sets are defined to indicate the appropriate relationships. The location modes of DEPT, ADDR, and PROJ permit one of two accesses: sequential or random by hashing. The location modes of EMPL, EDUC, and ASSN are VIA to achieve the desired clustering. Thus the abstract model, in its essence, can be implemented on network systems.

4.2. IMPLEMENTATION ON A HIERARCHICAL SYSTEM HMS)

Using IMS as a primary example of a hierarchical DBMS, the design of the abstract model can be largely achieved. The basic construct of IMS is a tree (hierarchy): both physical database descriptions (DBD) and logical database descriptions (LDB) are expressed as a collection of trees. The DBD is very similar to the abstract model in that the abstraction also calls for trees starting from the rooted entities. In DBD, the children segments are stored physically or logically next to the parent segments. The clustering can then be achieved by storing the children segments physically next to the parent segment.

The optimal design considered before can be represented in IMS by defining three DBDs corresponding to the rooted entities Department, Ad-dress, and Project (Figure 11). Within Department, EMPL is the child of DEPT, and EDUC and ASSN are children of EMPL. Further, there are logical connections between DBDs to represent the remaining relationships. The users will work with LDBs (LDB can be made from several DBDs). In fact, many LDBs would have to be defined to satisfy all of the queries.

4.3. IMPLEMENTATION ON COBOL-BASED FILE SYSTEM

The designs produced using the abstract design model and the interactive tool can also be implemented in a traditional non-DBMS file system environment. We illustrate how COBOL, file applications may implement the physical design. (Note that such applications abound in DP industry.) For the optimal design considered earlier, there will be three files declared in the DATA DIVISION of a COBOL, program corresponding to the three rooted entities, i.e., DEPARTMENT, ADDRESS, and PROJECT. In order to implement clustering, the DEPARTMENT record will have the EMPLOYEE entity as a repeating group field (using the OCCURS clause). In turn, the EMPLOYEE field would have EDUCATION and ASSIGNMENT as repeating group fields. In order to show the relationships between the three files, pointers will be included as fields in the record definitions. Depending on the degree of relationships, the pointers will be singular or repeating group fields. Since physical addresses would not normally be known, the pointers will have to be symbolic.

DBTG Implementation of Example Problem

```
001   SCHEME NAME IS PERSONNEL.
002   AREA NAME IS DATA-AREA-ONE.
003   AREA NAME IS DATA-AREA-TWO.
004   AREA NAME IS DATA-AREA-THREE.

005   RECORD NAME IS DEPT;
006       LOCATION MODE IS CALC HASH-DEPNO USING DPNO IN DEPT;
                                  O R
          LOCATION MODE IS DIRECT (SYSTEM/SEQUENTIAL);
007       WITHIN DATA-AREA-ONE;
008       IDENTIFIER IS DPNO IN DEPT.
009       02  DPNO                ;TYPE IS CHARACTER 5.
010       02  REST-OF-DEPT        ;TYPE IS CHARACTER 101.

011   RECORD NAME IS EMPL;
012       LOCATION MODE IS VIA DEPT-EMPL SET;
013       WITHIN DATA-AREA-ONE;
014       IDENTIFIER IS EMPLNO IN EMPL.
015       02  EMPLNO              ;TYPE IS CHARACTER 7.
016       02  REST-OF-EMPL        ;TYPE IS CHARACTER 61.

017   RECORD NAME IS ADDR;
018       LOCATION MODE IS CALC HASH-ADDRNO USING ADDRNO IN ADDR;
                                  O R
          LOCATION MODE IS DIRECT(SYSTEM/SEQUENTIAL)
019       WITHIN DATA-AREA-TWO;
020       IDENTIFIER IS ADDRNO IN ADDR.
021       02  ADDRNO              ;TYPE IS CHARACTER 8.
022       02  REST-OF-ADDR        TYPE IS CHARACTER 42.

023   RECORD NAME IS PROJ;
024       LOCATION MODE IS CALC HASH-PROJNO USING PROJNO IN PROJ;
                                  O R
          LOCATION MODE IS DIRECT(SYSTEM/SEQUENTIAL)
025       WITHIN DATA-AREA-THREE;
026       IDENTIFIER IS PROJNO IN PROJ.
027       02  PROJNO              ;TYPE IS CHARACTER 6.
028       02  REST-OF-PROJ        ;TYPE IS CHARACTER 104.

029   RECORD NAME IS EDUC;
030       LOCATION MODE IS VIA EMPL-EDUC SET;
031       WITHIN DATA-AREA-ONE;
032       IDENTIFIER IS EDUCNO IN EDUC.
033       02  EDUCNO              ;TYPE IS CHARACTER 3.
034       02  REST-OF-EDUC        ;TYPE IS CHARACTER 9.

035   RECORD NAME IS ASSN;
036       LOCATION MODE IS VIA EMPL-ASSN SET;
037       WITHIN DATA-AREA-ONE;
038       IDENTIFIER IS ASSNNO IN ASSN.
039       02  ASSNNO              ;TYPE IS CHARACTER 4.
040       02  REST-OF-ASSN        ;TYPE IS CHARACTER 11.
```

Fig. 10.

```
041         SET NAME IS DEPT-EMPL;
042         OWNER IS DEPT;
043             ORDER IS SORTED;
044         MEMBER IS EMPL;
045             INSERTION IS AUTOMATIC
046             RETENTION IS MANDATORY;
047             KEY IS ASCENDING EMPLNO IN EMPL
048               DUPLICATES ARE NOT ALLOWED
049               NULL IS NOT ALLOWED;
050             SET SELECTION IS THRU DEPT-EMPL OWNER
051               IDENTIFIED BY IDENTIFIER EMPLNO IN EMPL.
052     SET NAME IS EMPL-EDUC;
053         OWNER IS EMPL;
054         MEMBER IS EDUC;
                 —
055     SET NAME IS DEPT-MGR;
056         OWNER IS DEPT;
057         MEMBER IS EMPL;
                 —
058     SET NAME IS EMPLY_ASSN;
059         OWNER IS EMPL;
060         MEMBER IS ASSN;
                 —
061     SET NAME IS ADDR-DEPT;
062         OWNER IS ADDR;
063         MEMBER IS DEPT;
                 —
064     SET NAME IS ADDR-EMPL;
065         OWNER IS ADDR;
066         MEMBER IS EMPL;
                 —
067     SET NAME IS PROJ-MGR;
068         OWNER IS PROJ;
069         MEMBER IS EMPL;
                 —
070     SET NAME IS PROJ-ASSN;
071         OWNER IS PROJ;
072         MEMBER IS ASSN;
                 —
073     SET NAME IS SYSTEM-DEPT;
074         OWNER IS SYSTEM;
075         MEMBER IS DEPT;
076     SET NAME IS SYSTEM-ADDR;
077         OWNER IS SYSTEM;
078         MEMBER IS ADDR;
079     SET NAME IS SYSTEM-PROJ;
080         OWNER IS SYSTEM;
081         MEMBER IS PROJ;
```

Fig. 10. (*Continued*)

Note that in the nondatabase environment, the programmer will have to write his own "procedural" program for navigating through the multiple files and finding the related records from one file to another through the explicit use of pointers. Although the programming task in the non-DBMS environment may be harder, the same kind of efficiencies will be obtained by implementing the design obtained by using the DSS tool, as in the case of a DBMS implementation.

IMS Physical Database Description One

```
01    DBD NAME =      DEPTDB, ACCESS =HSAM (OR HISAM OR HDAM)

02    SEGM NAME =     DEPT, BYTES =106, FREQ =100, PTR =T,
                      PARENT =(ADDR,VIRTUAL,ADDRDB)
03    FIELD NAME =    (DPNO,SEQ,U), BYTES =5, START =1, TYPE =C
04    FIELD NAME =    REMDP, BYTES =101, START =6, TYPE =C

05    SEGM NAME =     EMPL, BYTES =68, FREQ =30, PTR =T,
                      PARENT =((DEPT),(ADDR,VIRTUAL,ADDRDB),
                                      (PROJ,VIRTUAL,PROJDB))
06    FIELD NAME =    (EMPNO,SEQ,U), BYTES =7, START =1, TYPE =C
07    FIELD NAME =    REMEMPL, BYTES =61, START =8, TYPE =C

08    SEGM NAME =     EDUC, BYTES =12, FREQ =2, PTR =T,
                      PARENT =EMPL
09    FIELD NAME =    (EDUCNO,SEQ,U), BYTES =3, START =1, TYPE =C
10    FIELD NAME =    REMEDUC, BYTES =9, START =4, TYPE =C

11    SEGM NAME =     ASSN, BYTES =15, FREQ =2, PTR =T,
                      PARENT =((EMPLY),(PROJ,VIRTUAL,PROJDB))
12    FIELD NAME =    (ASSNO,SEQ,U), BYTES =4, START =1, TYPE =C
13    FIELD NAME =    REMASSN, BYTES =11, START =5, TYPE =C
```

IMS Physical Database Description Two

```
01    DBD NAME =      ADDRDB, ACCESS =HSAM

02    SEGM NAME =     ADDR, BYTES =50, FREQ =3100, PTR =T
      LCHILD NAME =   (DEPT,DEPTDB)
      LCHILD NAME =   (EMPL,DEPTDB)
03    FIELD NAME =    (ADDRNO,SEQ,U), BYTES =8, START =1, TYPE =C
04    FIELD NAME =    REMADDR, BYTES =42, START =9, TYPE =C
```

IMS Physical Database Description Three

```
01    DBD NAME =      PROJDB, ACCESS =HSAM

02    SEGM NAME =     PROJ, BYTES =110, FREQ =200, PTR =T
      LCHILD NAME =   (EMPL,DEPTDB)
      LCHILD NAME =   (ASSN,DEPTDB)
03    FIELD NAME =    (PROJNO,SEQ,U), BYTES =6, START =1, TYPE =C
04    FIELD NAME =    REMPROJ, BYTES =104, START =7, TYPE =C
```

Fig. 11.

5. SUMMARY

Physical database design is a complex and demanding activity. As such, mathematical programming approaches to optimize the physical database have had only limited success. Moreover, the designer may have design preferences he may want to experiment with. In this paper, we have described an interactive DSS tool, which aides the database designer in the task of physical database design. The database design is accomplished in the context of a high-level abstract model which is capable of implementation in a variety of DBMSs. The interactive tool not only lets the designer experiment with his own designs, but also provides several heuristic procedures to enable the generation of many good designs. Another feature of the DSS tool is exhaustive enumeration of physical designs, which may be used for small problems. Finally, the paper demonstrates how the physical design selected using the abstract model and the interactive tool may be implemented on several systems. Examples on a network system, hierarchical system and file system were given.

REFERENCES

1. H. D. Anderson and P. B. Berra, Minimum cost selection of secondary indexes for formatted files, *ACM TODS* 2:68-90 (1977).

2. D. S. Batory, Modeling the storage architectures of commercial database systems, *ACM TODS* 10(4):463--528 (Dec. 1985).
3. D. S. Batory and C. C. Gotlieb, A unifying model of physical databases, *ACM TODS* 7(4):509-539 (Dec. 1982).
4. BCS/CODASYL DDLC Data Base Administration Working Group, Draft Spec. of Data Storage Description Language, 1978, Appendix.
5. J. V. Carlis, Investigation in the modeling and design of large, logically complex, multi-user database, Ph.D. thesis, University of Minnesota, 1980.
6. J. V. Carlis and S. T. March, Computer-aided physical database design methodology, *Comput. Performance:198-214* (Dec. 1983).
7. J. V. Carlis, S. T. March, and G. W. Dickson, Physical database design: a DSS approach. *Inform. Manage.* 6:211-224 (1983).
8. D. D. Chamberlin et al., History and evaluation of system R, *Commun. ACM:* 632-646 (Oct. 1981).
9. C. J. Date, *An Introduction to Database Systems,* 4th ed., Addison-Wesley, Reading, Mass., 1985, Vol. 1.
10. J. R. Evans, Toward a framework for heuristic optimization, in *Proceedings AIIE Annual Conference,* Spring 1979.
11. T. J. Gambino and R. Gerritsen, A data base design decision support system, in *Proceedings of the 3rd International Conference on VLDB,* Tokyo, Japan, 1977.
12. R. Gerritsen, Cost effective database design: an integrated model, in *Proceedings of the Conference on Advanced Information System Development Techniques Symposium,* April 1977.
13. A. Guttman and M. Stonebroker, Using a relational database management system for computer aided design data, *Bull. IEEE Comput. Soc. Tech. Commun. Database Eng.* 21-28 (Jun. 1982).
14. M. Hammer and A. Chan, Index selection in a self-adaptive data base management system, in *Proceedings of the ACM SIGMOD,* 1976.
15. M. Hammer and B. Niamir, A heuristic approach to attribute partitioning, in *Proceedings ACM SIGMOD,* Boston, Mass., 1979.
16. J. A. Hoffer and A. Kovacevic, Optimal performance of inverted files, *Operations Res.* 30(2):336-354 (Mar.-Apr. 1982).
17. J. A. Hoffer and D. G. Severance, The use of cluster analysis in physical database design, in *Proceedings of the 1st International Conference on VLDB,* September 1975.
18. D. Hsiao and F. Harary, A formal system for information retrieval from files, *Commun. ACM* 13(2):67-73 (Feb. 1970).
19. H. K. Jain, A comprehensive model for the storage structure design of CODASYL databases, *Inform. Sys.* 9(3/4):217-230 (1984).
20. H. K. Jain and J. R. Krobock, Computer aided system for the database storage structure design, *Inform. Manage.* 6(6):337-349 (Dec. 1983).
21. R. H. Katz and E. Wong, Resolving conflicts in global storage design through replication, *ACM TODS* 8(1):110-135 (Mar. 1983).
22. D. M. Kroenke, *Database Processing,* 2nd ed., SRA, Chicago, 1983.
23. S. T. March, Techniques for record structuring, *ACM Comput.* Sum.:45-80 (Mar. 1983).
24. S. T. March and D. G. Severance, The determination of efficient record segmentation and blocking factors for shared data files, *ACM TODS* 2(3):279-296 (Sept. 1977).
25. P. Palvia, An analytical investigation into record structuring and physical database design, Ph.D. thesis, University of Minnesota, 1984.
26. P. Palvia, How sensitive is the physical database design?-results of experimental investigation, *Proceedings of the NCC,* June 1987.
27. P. Palvia, Heuristic optimization of physical databases: using a generic and abstract design model, *Decision Sci.* 19(3):564-579 (Summer 1988).
28. M. Schkolnick, A clustering algorithm for hierarchical structures, *ACM TODS* 2(1):27-44 (Mar. 1977).
29. M. Schkolnick, A survey of physical database design methodology and techniques, in *Proceedings of the 4th International Conference on Very Large Databases,* West Berlin, Germany, 1978.
30. D. G. Severance, A parametric model of alternative file structures, *Inform'. Syst.* 1(2):31-55 (1975).

31. K. Y. Whang, G. Weiderhold, and D. Sagalowicz, Physical design of network databases using the property of separability, in *Proceedings of the 8th International Conference on Very Large Databases*, September 1982.
32. S. B. Yao, An attribute based model for database access cost analysis, *ACM TODS* 2(1):45-67 (Mar. 1977).
33. S. H. Zanakis and J. R. Evans, Heuristic "optimization": why, when, and how to use it, *Interfaces* 11(5):84-91 (Oct. 1981).