The Database Designer's Workbench*

RICHARD E. COBB

JAMES P. FRY
and
TOBY J. TEOREY

Information Systems Research Group, Graduate School of Business Administration, University of Michigan, Ann Arbor, Michigan, 48109

Communicated by K. S. Fu

ABSTRACT

The Database Designer's Workbench is a graphics-oriented decision support system for database design, providing designers with a convenient environment for specifying database structures and experimenting with different design strategies. A prototype version has been implemented on the Honeywell Multics system using the IBM Personal Computer as the interface. Tools for conceptual design and distributed data allocation are provided. This paper describes the Workbench design objectives, system architecture, and current status.

1. INTRODUCTION

The Database Designer's Workbench (or "Workbench") is a graphics-oriented decision-support system to assist with the design of all aspects of a computerized database, from the initial specification of the system's requirements through its final physical structure. It provides a wide variety of design aids, or "tools," for designers to explore many design alternatives and to evaluate them precisely. These tools are presented in a homogeneous, graphically oriented environment that allows a designer to use familiar representations, store incomplete designs, progress smoothly from one design phase to the next, and iterate over previous design stages. The Workbench is therefore a real asset to the database design practitioner, seeking to improve productivity and

*This research was supported by Rome Air Development Center (USAF) Contract # F30602-82-C-0003.

the quality of design. The Workbench is also of value to researchers interested in improving the database design process because it monitors and records individual designer use.

The Workbench supports a stepwise and iterative database structure refinement methodology described by Teorey and Fry [17, 18]. The methodology is based on well-established system design principles, specific requirements, and the environmental constraints of database systems [12]. If database design steps were completely independent of each other, a single pass through them would result in an optimal design. There are many dependencies, however, some resulting from the biases of the designers themselves and others inherent in the design process. Hence, iteration is a necessary part of the design process. Iteration within a major step may be necessary to optimize over one or more independent variables. Iteration across major steps is required when certain requirements are not met by any of the alternative designs at a particular step, or if new requirements are added that require redesign. As an example, if it is found that candidate physical database structures cannot meet response-time criteria, new logical database structures, or even new conceptual information structures may have to be considered (see Figure 1). One of the main purposes of the Workbench is to provide design tools that simplify the process of redesign and evaluation.

The Workbench provides several advantages over both manual methods of database design and presently existing automated tools. It enables researchers and database administrators to design, experiment with, and use new tools of their own. It encourages this by relieving the designers of some of the details involved in developing user interfaces, input/output of design parameters, and storage specifications. The graphic interface allows the designers to see the various structures they are considering for their database throughout the entire design process. Since database structures are easily modified graphically, designers are encouraged to produce and consider more candidate structures. Also, when a designer progresses from one design phase to the next and is faced with a set of possible structures to choose from, the system's graphical nature can significantly aid in the heuristics of selection.

1.1. RELATED RESEARCH

The Database Designer's Workbench grew out of an earlier project of the Information System Research Group at the University of Michigan known as the "Interactive Database Design Methodology" (IDDM) [20]. The IDDM shared a common goal with the Workbench, that of assisting a user in the design and analysis of database structures. It made use of a similar "tool" paradigm to the Workbench, wherein design aids are incorporated into the system as passive elements to be selected at the discretion of a designer.

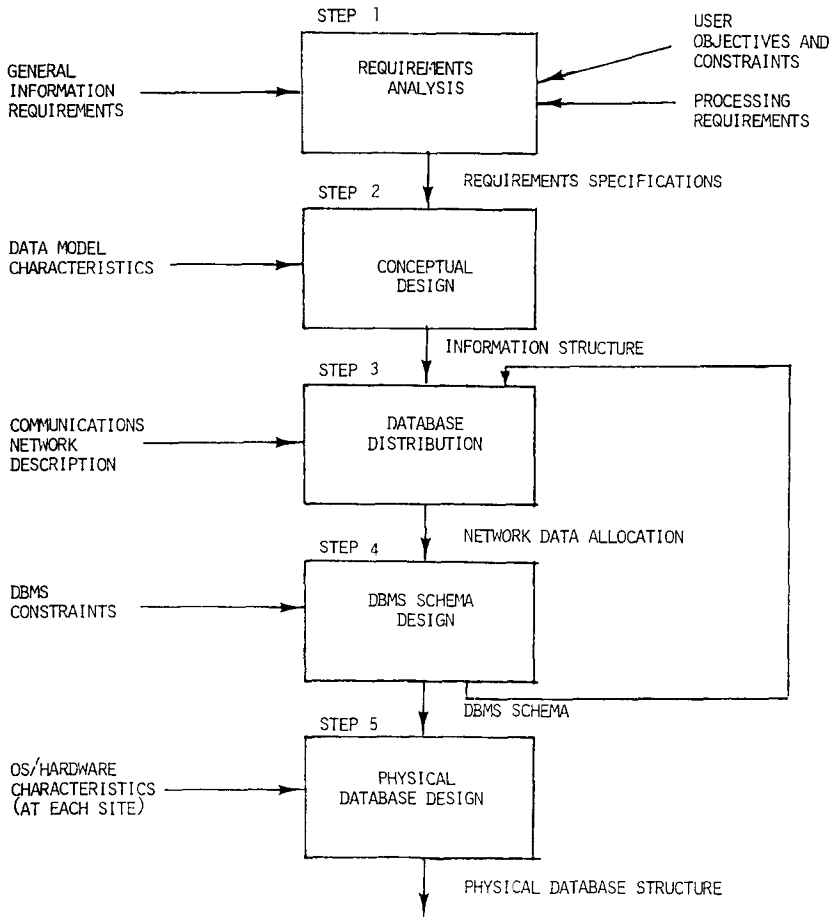

Fig. 1. Database structure refinement methodology.

The IDDM, however, suffered from three major difficulties. The first of these was the lack of a coherent, user-friendly interface. The project's major emphasis was the selection and implementation of important database models and algorithms rather than in the ultimate utility of these design aids to the designer. The result was a set of tools which were overly general and unwieldy. The Workbench has therefore broken up these tools into smaller (i.e., more specific), more manageable ones. Second, the IDDM attempted to integrate existing, heterogeneous models and evaluators. The wide variety of formalisms and representations used by these models made it very difficult for a user to coordinate their use (and even more difficult for the IDDM researchers to implement them).

The third difficulty that the IDDM encountered arose when a designer attempted to use the results from one design stage as input into the next. The linkages between the various design stages were weak (again because they had to be added on instead of being built in), and the utilities for storing intermediate design results were quite primitive. This made it very difficult to achieve any sort of complete design process or methodology.

A number of related research activities exist outside the University of Michigan. At the University of Toronto, Lochovsky has developed a design tool for entity-relationship diagrams [4]. The work in Sweden on the CS4 system is more advanced, but is limited to the beginning designing steps [1]. Data Designer, offered commercially, has only a single design tool [10], although several extensions are planned. Most recently the ISDOS/PRISE project at the University of Michigan has developed a "View Integration System" [6]. It is significant in that it integrates a requirement language (PSL) and data dictionary facility with a design tool, the "View Integrator."

1.2. WORKBENCH APPROACH

The Workbench was developed from a quite different perspective than the other efforts: rather than just implementing a single model or beginning with a number of diverse models and then trying to link them, the Workbench first establishes a generic model of the design process. It incorporates a single, graphically intuitive representation for the interface, and establishes a framework for each of the models. As a consequence two interesting results have emerged. The system interface is homogeneous from the user's point of view—because the interface has been built in, rather than added on. Second, the models or tools have been broken up into smaller, more manageable modules.

To address the problem of linkages between the design stages, the Workbench incorporates a design database for maintaining all of the information used and generated by the various design tools. (This "database of designs" is to be distinguished from the "target databases that may be designed" using the Workbench.) Secondly, it is now possible to link the tools into methodologies: strategies for the design of entire databases from requirement specifications to physical structures [18]. These are desirable when the Workbench is used by a novice designer, or to provide a preliminary, "rough cut" design to an expert.

1.3. DATABASE DESIGN AS A PROBLEM-SOLVING ACTIVITY

One of the things learned from the Workbench prototype experience is that both the Workbench and its users require a more complete view of the database design process. This allows the individual design tools to be integrated into a larger framework, as well as highlighting those "leverage points" in the design process at which the Workbench can provide most assistance.

The metaphor adopted was the decomposition of problem solving [16] into iterations on three general activities: generating alternatives, evaluating alternatives, and selecting the "best" alternative. The choices made in each of these steps eventually form a "path" to the solution. The Workbench logs all interaction which moves the designer along this path, including backtracks, etc. This provides database design researchers with some data on the design process itself, which could lead eventually to both better methodologies and expert systems for database design.

The database structure refinement methodology divides the problem-solving activities into five basic steps shown in Figure 1. Each step represents a complete design phase with given requirements, constraints, design techniques, evaluation criteria, and a design product.

Requirements Analysis (Step 1)

The designer begins with some "real-world situation" (i.e., an organization full of people, processes, goals, constraints, data, etc.) which must be analyzed in terms of the requirements it places upon its information system. Normally these requirements are solicited through personal interviews with managers and employees from many levels within the organization and then documented in a requirements specification language. This documentation should identify each process, data associated with the process, and relationships between processes as well as specific performance, security, reliability, and "political" constraints.

Conceptual Design: View Modeling and Integration (Step 2)

Perhaps the most difficult and least understood task is to design an information structure which satisfies the requirements imposed by the last step. This information structure will form an initial, high-level representation of the database that is independent of the DBMS used to implement it. There are two basic approaches to the development of such a structure. The first is a top-down approach in which semantic assertions about data entities, attributes of entities, and relationships between entities are made [5]. The second method is bottom-up: data elements are analyzed in terms of the individual associations in which they are involved. These "natural" views of the information structure are usually preferable to "usage" views, because the semantic relationships the natural views capture are less transient than the activities on which the usage views are based. Unfortunately, it is usually much more difficult to ferret out fundamental, semantic relationships, especially if the enterprise is large or the analyst is unfamiliar with it; observing which activities use which data is much more straightforward.

Database Distribution (Step 3)

Once an adequate conceptual model has been created, the design progresses into decisions about which system to use to implement the database. In a distributed computing environment, this means choosing logical fragments (of database relations or files) for distribution, selecting nodes for individual fragments and programs, and determining the necessary concurrency and synchronization techniques. The result is a specification of the individual information to be provided at each node; this forms the basis for the design of system-specific database schemas.

DBMS Schema Design: (Step 4)

The next task facing the database designer is to combine the general information structure developed in the previous stage with usage patterns in a particular application. Those assertions which can be made about fundamental aspects of the information structure can be used to constrain the range of structures suggested by correct usage patterns; alternatively, the usage view can be used to confirm that the natural view is feasible. In any case, the result is a local view of the information structure for each application; these must then be integrated into a single, global view, with all inconsistencies and redundancies identified and resolved.

The information structure is thus mapped into the DBMS's data-definition language (DDL), and the transactions into its data-manipulation language (DML). These transformations are usually quite straightforward, with the main difficulties arising because of DBMS idiosyncracies. However, if the conceptual-structure formalism and logical DBMS data model selected are very dissimilar, this too can make the task of implementing the structure in a DBMS very difficult.

Physical Database Design (Step 5)

Finally, the DBMS schema and transactions must be refined again, based this time upon the specifics of the computer hardware and operating system used at each node in the network, as well as upon usage patterns. It is at this stage that the designer incorporates the physical access paths, data compression techniques, device assignments, block sizes, etc. into the design. The result is a complete database specification.

2. SYSTEM ARCHITECTURE

A quite different perspective on the Workbench is afforded by considering the actual software underlying the abstract functional specifications mentioned heretofore. This section will briefly review the system architecture portrayed in Figure 2.

The user communicates with the system through a combination of commands, menu selections, and cursor positioning. The *screen command interface* receives each of these optional forms, uses contextual information contained in the screen display, and forms a complete command in a common format. This command is sent to the *dispatcher*, which logs the interaction (as part of the monitoring of the Workbench's use) and uses it to activate a tool or series of tools.

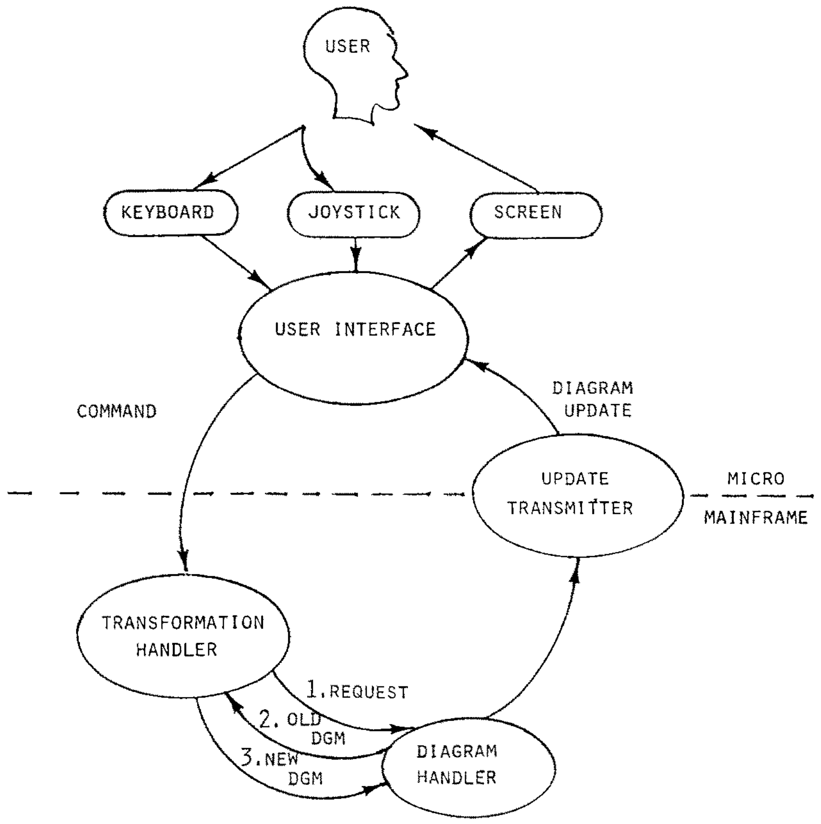

Fig. 2. The Workbench architecture.

These tools vary widely; what they have in common is that they either alter the design diagram currently on the screen or evaluate one of its properties. In either case, the tool must make use of the information contained in the *design database*; interaction with this database is done through the *diagram handler*.

The diagram handler is also responsible for forwarding any changes in the design to *screen update*, which transforms these changes in the design's diagram. It has proven useful to maintain a separate *screen database* of the objects displayed on the screen; this database is manipulated by the *screen handler*. The *drawer* translates these general specifications for changes to the diagram into instructions for the actual display device. Note that only the screen command interface and the drawer are device-dependent. A more detailed discussion of Workbench components is given in [8].

When using the Workbench, the graphic screen will appear as in Figure 3. The line across the bottom of the screen is a delimiter line, reserving the bottom portion for textually oriented information. Most commands available with the screen command interface (all except those which modify only the display) are available using the textual command interface. The action of the graphic interface is to translate a graphic command to a textual one, then interpret the response as commands to draw particular objects. An example is shown in Figure 4. Specific features included in the user command structure include the ability to create, modify, copy, save, restore, or delete a design. Most of these

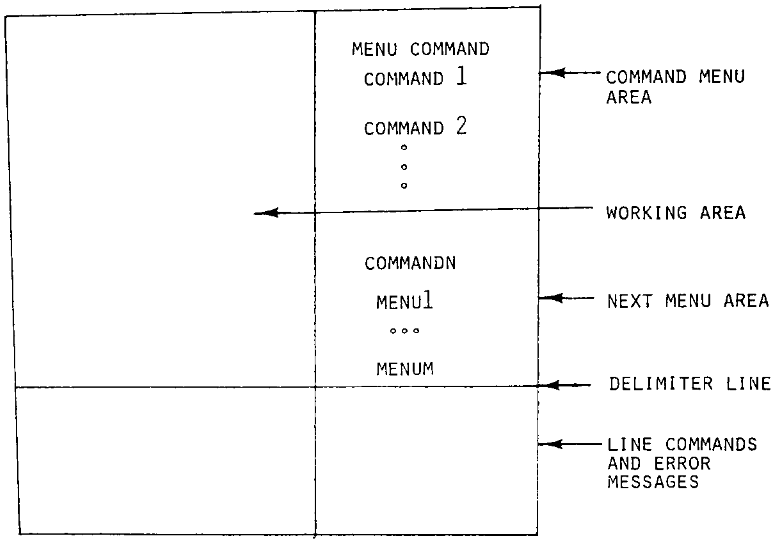

Fig. 3. The form of the Workbench screen.

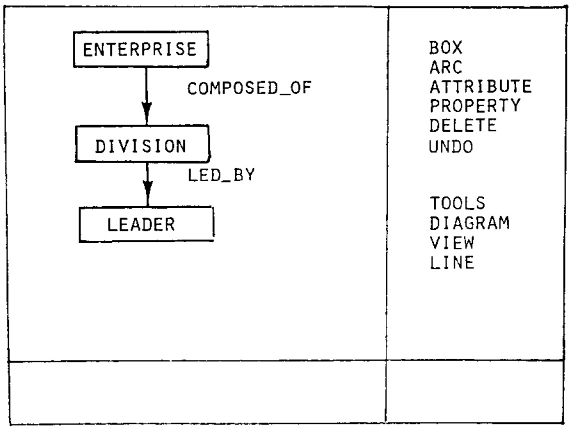

Fig. 4. An example screen.

functions are also applicable to individual data objects (entities, attributes, records, etc.), relationships, and user transactions.

Specifically graphic functions, in particular the handling of large diagrams, are provided by the screen handler. The Workbench can display a diagram at either of two levels of detail: the "global" view, which contains only the boxes and arcs within a diagram, and the "local" view, in which a central "box" (representing a record class, relation, etc.) has all its information displayed (attributes, performance specifications). This box is displayed at the center of the screen, and all those boxes which are either connected to it with an arc or geographically close (determined when a box is first added to a diagram) are clustered around it. An attempt is made to keep this local view as consistent with the topography of the global view as possible while placing all the relevant information on the screen.

3. WORKBENCH IMPLEMENTATION

3.1. *PROTOTYPE WORKBENCH*

The prototype Workbench implemented on MULTICS adheres closely to the configuration given in Figure 2. This working prototype uses an IBM PC as its graphics input/output device and command level processor; a Honeywell Multics system contains the algorithms, tools, and diagram handler. The command-level processing portion of the software is written in PASCAL. The diagram handler and several tools are written in LISP; other tools are in PASCAL. Normal "command line" interaction is possible through any Multics terminal [7].

3.2. *TOOL IMPLEMENTATIONS*

Two quite different tools have been developed for the Workbench: the CBNF "functional dependence" methodology for design of normalized relations, and an optimal data-allocation strategy for distributed databases based on the methodology of Morgan and Levin [15].

CBNF is a transformational tool which synthesizes normalized relations from a set of functional dependencies which represent the natural associations among data elements defined in the enterprise model. It is used in the conceptual-design step. The output relations are in third normal form, excepting those attributes which were found to be reflexively dependent on each other. Any two such attributes are considered to be isomorphic, and thus are treated together as a single attribute in the normalization algorithm. Such attributes are flagged in the resultant normalized relational schema; they point out the simplest sort of cyclic dependency [14].

The data allocation for distributed systems (DADS) tool implements an algorithm for evaluating and optimizing program and data allocation across a distributed database [15, 19]. The tool parameters include information about network topology (processing nodes and communication links), data volume, programs, and higher-level user transactions. When run in evaluation mode, the tool produces cost and delay-time information for a given program and data-allocation scheme over a set of user transactions. Costs are broken down into storage costs, local-processing costs, and communication costs. When run in optimization mode, the minimum-cost configuration for program and data allocation is determined.

3.3. *FUTURE TOOLS*

Two additional tools have been selected as the first extensions to the Workbench tool library: the logical record access (LRA) evaluator [17, 18], which is used in DBMS schema design, and the usage-dependency model [13], which can be used in either conceptual design or DBMS schema design.

The LRA evaluator estimates the number of occurrences of each record type retrieved in the course of executing a database application. It is used to evaluate the relative efficiency of candidate DBMS schemas. Experience has shown that while the logical record access metric is limited as a predictor of real database performance (it does not take into account physical parameters such as block size or randomness of record accesses), it can detect large differences in efficiency between candidate schemas. More importantly, it can be used to compute transport volume, i.e., the total number of bytes transferred for a particular application. Transport volume is known to be a very good predictor of physical data transfer. The LRA evaluator will later be extended to include physical parameters needed to accurately predict the I/O time to perform an application [18].

The usage-dependency model is an extension of the functional-dependency model of the relational database theory that specifies the probability of joint data usage based on process and data correspondence identified in the user requirements. The combined application of usage dependency, which is a process-oriented view of data correspondence, and functional dependency, which is a nonprocessing or natural view of data correspondence, can produce a normalized database structure that is efficient for current processing requirements as well as flexible for future usage. The database structure can be automatically generated by an algorithm of polynomial complexity.

4. SUMMARY AND CONCLUSIONS

Central to the evolution of the Database Designer's Workbench is a fundamental system paradigm which has guided its development. This model consists of four components:

(1) a methodology,
(2) user interface,
(3) design tools, and
(4) Workbench environment.

Recent results indicate that this model provides a robust approach to the implementation of a centralized and/or distributed database design methodology as well as a supporting environment for design tools.

4.1. METHODOLOGY

The notion of a database design methodology is that it is useful to break up this large, difficult task into a number of smaller design steps. The Workbench facilitates this decomposition, and this has allowed us to focus our energies on those "leverage points" in the design process at which the system can provide the designer with the most assistance. While the current "top-down" methodology which begins with requirements analysis and ends with physical design remains a valid high-level decomposition of the problem, it is useful and timely to begin considering alternative approaches in the individual phases. For example, tools for analyzing existing database applications should be provided for DBMS schema design.

4.2. USER INTERFACE

The defining characteristics of the Workbench's user interface are, first, its graphical nature and, second, the fact that the representations chosen are ones already familiar to the database designer. Possible extensions to the Workbench interface are the representation of higher-level enterprise models and "forms" for representing user requirements.

4.3. DESIGN TOOLS

The heart of the workbench concept is the set of tools that the system provides. Some of these tools help a designer generate possible design alternatives, while others evaluate designs that the designer has generated. As database research continues to improve our understanding of the relevant issues, algorithms appear which can be used in both of these ways.

Candidates for the tool repertoire were and will continue to be selected according to the following criteria:

(1) Design tools should be small, addressing particular stages of the design process, rather than large, ad hoc ones encompassing the entire design process.

(2) They should provide maximum "leverage" i.e., high payoff to real-life problems.

(3) Emphasis will be placed on those tools which help with the early conceptual and logical phases, as they are most needed and most likely to remain valuable as computing hardware evolves.

4.4. WORKBENCH ENVIRONMENT

One of the major advantages of the Workbench concept is the degree of integration provided between the many disparate design tools. This is due to the perspective offered by a complete design methodology. Also, because the most dynamic element of the Workbench—new tools—has been isolated as a separate component of the system, it can adapt gracefully as advances become available.

As the designer progresses through the design process, there are a number of intermediate structures generated. Usually it is necessary to iterate through previous stages several times before a final design is reached. The existence of a design database has made organizing and modifying these intermediate results much easier. Future activities involve the experimentation with both novice and expert database designers to evaluate Workbench responsiveness, clarity, and flexibility.

The authors gratefully acknowledge the technical contributions from Rik Belew, Alberto Garcia, Paul Helman, Leo Hourvitz, Marilyn Mantei, Deborah Meredith, Shifrah Nenner, Wayne Sawdon, J. B. Smelcer, Peter Stevenson, Paul Stutler, Cole Whiteman, and Dawn Zelinski.

REFERENCES

1. S. Berild et al., *CS4: An Introduction to Associative Databases and the CS4-System*, Chartwell-Bratt, Bromley, Kent, G.B., 1981.
2. P. A. Bernstein, Synthesizing third normal form relations from functional dependencies, *ACM Trans. Database Systems* 1(4):277–298 (1976).
3. C. Berri and P. A. Bernstein, Computational problems related to the design of normal form relational schemas, *ACM Trans. Database Systems* 4(1):30–59 (Mar. 1979).
4. E. P. F. Chan and F. H. Lochovsky, A graphical data base design aid using the entity-relationship model, in *Proceedings of the International Conference on Entity-Relationship Approach to Systems Analysis and Design* (P. P. Chen, Ed.), 1979.
5. P. Chen, *The Entity-Relationship Approach to Logical Data Base Design*, Q.E.D. Monograph Ser., Wellesley, Mass., 1977.
6. W. P. Chiang, E. Basar, C. Lien, and D. Teichroew, Data modelling with PSL/PSA: The view integration system (VIS)," PRISE Ref. M0509-0, Univ. of Michigan, July 1983.
7. R. Cobb, Database Designer's Workbench user's guide, Working Paper 81 DE 1.18, Information Systems Research Group, Univ. of Michigan, Oct. 1981.
8. R. Cobb, Data Designer's Workbench design specifications, Working Paper 81 DE 1.11, Information Systems Research Group, Graduate School of Business Administration, Univ. of Michigan, Oct. 1981.
9. E. F. Codd, A relational model of data for large shared data banks, *Comm. ACM* 13(6):377–387 (1970).
10. *Data Designer*, Database Design, Inc., Ann Arbor, Mich., 1981.
11. C. J. Date, *An Introduction to Database Systems*, 3rd ed., Addision-Wesley, Reading, Mass., 1983, Vols. 1, 2.
12. J. P. Fry and E. A. Sibley, Evolution of database management systems, *ACM Comput. Surv.* 8:7–42 (1 Mar. 1976).
13. E. Hevia and T. J. Teorey, Usage dependency model for logical database design, Information System Research Group Technical Report 83 DE 22, Univ. of Michigan, Sept. 1983.
14. K. Laver and M. H. Graham, Functional dependencies on cyclic database schemes, in *Proceedings of the 1983 ACM-SIGMOD Database Week*, San Jose, 23–26 May 1983, pp. 79–91.
15. H. L. Morgan and H. D. Levin, Optimal program and data locations in computer networks, *Comm. ACM* 20(5):315–322 (1977).
16. N. J. Nilsson, *Problem Solving Methods in Artificial Intelligence*, McGraw-Hill, New York, 1971.
17. T. J. Teorey and J. P. Fry, The logical record access approach to database design, *ACM Comput. Surv.* 12(2):179–211 (June 1980); Corrigendum, 12(4):465 (Dec. 1980).
18. T. J. Teorey and J. P. Fry, *Design of Database Structures*, Prentice-Hall, Englewood Cliffs, N.J., 1982.
19. A. Umar, Analysis of distributed database systems Ph.D. Dissertation (in progress), Univ. of Michigan.
20. M. E. Wilens, R. A. Volz, and J. P. Fry, Interactive database design methodology, Tech. Report 78 DE 13, Database Systems Research Group, Graduate School of Business Administration, Univ. of Michigan, Ann Arbor, Apr. 1978.