C. P. Wang
H. H. Wedekind

Segment Synthesis in Logical Data Base Design

Abstract: Identification and representation of entities and their relationships relevant to an application are some of the key problems in logical data base design. This paper presents an approach to synthesizing logical segments that are representations of such entities and relationships. The major steps in this design are 1) collect all the pertinent functional relations in the application domain; 2) remove redundant relations to obtain a minimal covering set; 3) minimize the number of relations in the covering set to obtain an optimal set of relations in the third normal form; and 4) combine relations into logical segments according to prescribed performance requirements and projected information maintenance activities. Synthesis of logical segments for an airline reservations application is used as an illustrative example.

Introduction

At the current state-of-the-art, the methods used in the design of data base systems are essentially trial-and-error, supported by neither a scientific foundation nor an engineering discipline. The haphazard approach to design frequently leads to inflexible solutions that do not meet the prescribed requirements. Costly remedial measures often produce more delay in operation without a tangible improvement. Much of the existing information on system design is presented in the form of case studies [1]. These cases do provide valuable insight, but they can hardly be adequate substitutes for a systematic design discipline.

Before contemplating a basis for design, an appropriate conceptual framework of multilevel abstractions must be established for an information system. At each level, design issues and parameters can be localized with minimal dependency on parameters of adjacent levels. It is generally accepted that there should be at least three basic levels of system abstractions: 1) the user's level of abstraction, viewed by individual users or application programs; 2) the system logical level of abstraction, combining the views of many applications into a centrally controlled and maintained logical data base, with provisions for data sharing and security; and 3) the system physical level of abstraction, reflecting all the implementational idiosyncracies of a system. Several recent data base system proposals [2-4] have provisions for a multilevel architecture.

In this paper, we consider only the design methods to be employed at the user's level of system abstraction. Specifically, we consider the description of an application problem in quantitative terms and the synthesis of logical segments for the data base. Of particular interest is the set of primitive functional relations that represent all the relevant entities, their properties, and their relationships in an application. This set of functional relations can be analyzed to remove redundancy and then combined into an optimal set of relations in the *third normal form* [5]. By taking into account the prescribed performance requirements and the insertion, deletion and update activities of the application [6], the optimal collection of relations can be further combined into logical segments of the data base. An airline reservations application is used as an example to elucidate the design process.

Defining the problem domain for an application

Consider the object *automobile*, a well known entity. To a person in the state Motor Vehicle Registration Department, the notions of *year, make, model, vehicle ID no., name legal owner* are clearly important attributes of this entity. To an auto repair shop, other attributes, such as *engine model, brake type, type of transmission*, and *tire type and size*, are much more relevant properties. The choice of an entity type is not always a straight forward operation. As an entity, an automobile has many components, such as engine, brake, transmission, etc. Each component, for example the engine, may be considered an entity type in its own right. A component can further be decomposed into subcomponents, all of them may be considered as distinct entity types. It is probably futile to seek a set of primitive concepts as the building blocks for other concepts and objects simply because knowledge is open-ended. On the other hand, it is conceivable that within a well defined problem domain, the role of every concept can be identified. The

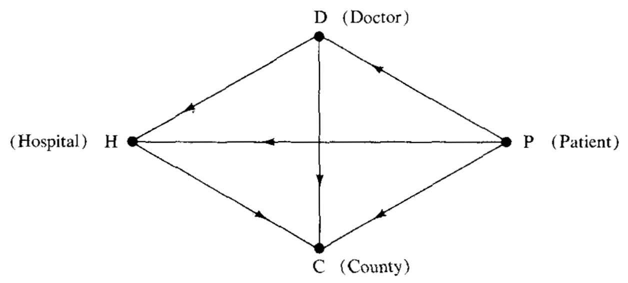

Figure 1 Graphical representation of functional relations

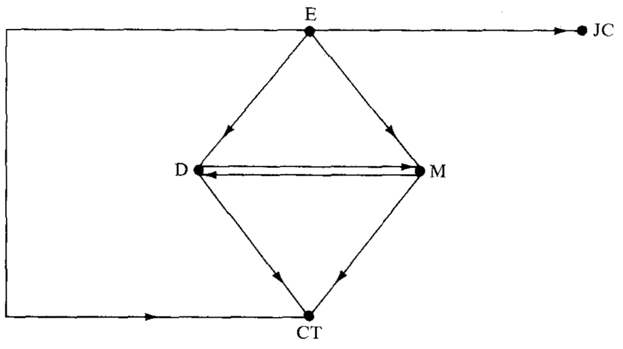

Figure 2 An example of functional relations between employees and their departments.

world as viewed within the boundary of this problem domain, called the mini-world, is surely much more restrictive than the world of our total knowledge.

In a mini-world of application, it may be reasonable to enumerate all the important facts and concepts. There are several established mini-world models, such as the block's world for robot research [7], the lunar rock world [8], the driver's world for natural language question-answering systems [9], the business-oriented applications [10], etc. The evidence indicates that a reasonably complicated application can be adequately described by several thousand facts, a magnitude that is amenable to computerized handling. The World-of-Business model currently being pursued by the automatic programming group at MIT project MAC [11] and the Business Definition System pursued by the automatic programming project at IBM Research, Yorktown Heights [12] are examples based on such a premise.

Representations of information objects

The idea of using sets to represent information objects can be traced back to 1962 in the work of information algebra [13]. More recent work on relational data bases [2] established the relational representation of entities and made it possible for the set algebra to bear on the manipulation of information entities. Decomposition of relations by means of more primitive functional relations between attribute domains has led to the definitions of three normal forms [5]. Even in a well defined problem domain, due to the inherent redundant nature of the descriptive facts about that mini-world, it might still be difficult to specify relations directly. A plausible alternative is to extract exhaustively all the relationships from a set of descriptive facts and then subject the collection of relationships to analysis to eliminate redundancy and check consistency before synthesizing them into relations. As indicated, the number of distinct facts may be in the thousands. It is unrealistic to expect that all the relationships can be extracted completely and accurately in one step. A more rational approach is to iterate the analysis process and provide intermediate feedback to reconcile incomplete and inconsistent relationships.

• *Functional relations*

Given a collection of n attribute domains, one can define a relation whose members are n-tuples by taking one element from each attribute domain. One can also analyze the functional associations between any two attribute domains. The notion of *functional relation* has been utilized by Codd [5] to define the second and third normal forms of relational representation of data bases and be Delobel and Casey [14] to decompose a large collection of attribute domains into files. The basic notion of a functional relation between two groups of attributes A_1, $A_2, \cdots, A_j$ and $B_1, B_2, \cdots, B_k$ can be defined as follows: there exists a function f such that $f(A_1, A_2, \cdots A_j) = (B_1, B_2, \cdots, B_k)$. The notation $\rightarrow$ is used to denote a functional relation, e.g., $A_1, A_2, \cdots, A_j \rightarrow B_1, B_2, \cdots, B_k$. Such a relation is called an *elementary* functional relation if no proper subset of attributes $A_u, A_v, \cdots, A_w \subset A_1, A_2, \cdots, A_j$ can be found such that $A_u, A_v, \cdots, A_w \rightarrow B_1, B_2, \cdots, B_k$ is also a functional relation. If two attribute domains X and Y are related in a one-to-one manner, this association can also be expressed as functional relation in both directions $X \rightarrow Y$ and $Y \rightarrow X$. The properties of functional relations have been studied elsewhere [14] and are not repeated here.

From a given set of functional relations, the transitivity enables one to derive all the possible functional relations. Thus, if $A \rightarrow B$ and $B \rightarrow C$ are functional relations, so is $A \rightarrow C$. The set of all possible functional relations is known as the *transitive closure*. To represent information, the transitive closure is redundant because some members are derivable from others. The notion of *minimal cover* is a minimal set of relations from which the *closure* can be derived [15]. For a given set of relations, there may be several minimal covers. Each is sufficient to derive the closure. From a representation point of view, a minimal cover is a logical choice for non-redundant representation of information.

To illustrate the concept of minimum cover, consider the set of functional relations represented by the graph in Fig. 1, which depict the associations among *patients,* *doctors, hospitals* and *counties*. Here, we assume that every patient has only one doctor, every doctor can practice in only one hospital and a hospital is located in a county. The functional relations $P \rightarrow D$, $D \rightarrow H$, and $H \rightarrow C$ are primitive. The functional relations $P \rightarrow H$, $P \rightarrow C$, and $D \rightarrow C$ are transitive and are derivable from the three primitive relations. In this case, the minimal cover is unique, consisting of three arcs $P \rightarrow D$, $D \rightarrow H$, and $H \rightarrow C$.

As a second example, consider a set of functional relations pertaining to employees and their departments as depicted in Fig. 2. The example was taken from reference [5]. Let E designate employee; JC, employee job code; D, department of employee; M, manager of employee; and CT, contract type. Assume further that each employee is given only one job code and assigned to one department. Each department has only one manager and is working on either defense related contracts or non-defense related contracts, but not on both. The graph has a cycle incident on the vertices D and M. The family of minimal covers is given in Fig. 3.

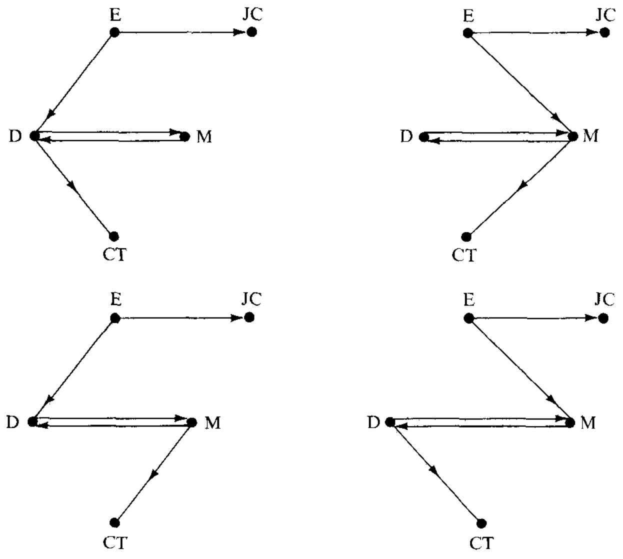

Figure 3 Four minimal covers for the example given in Fig. 2.

• *Minimal cover algorithm*

For a given set of relations it is possible to provide a general algorithm to derive all the minimal cover sets. The following algorithm was proposed by Delobel [16].

Let T denote the transitive composition of two functional relations. Let r_1 denote $A \rightarrow B$, r_2 denote $B \rightarrow C$, and r_3 denote $A \rightarrow C$. Then we can write $T(r_1, r_2) = r_3$. We designate the set of minimal covers by $cv\{s_i^m\}$.

Step 1 From the set of functional relations $S^k = \{r_1, r_2, \cdots, r_k\}$ remove an element r_p if there exist $r_i, r_j, r_p \in S^k$ such that $T(r_i, r_j) = r_p$. Call the remaining set S_p^{k-1}. If no element can be removed from S^k, place S^k in the set CV and terminate. In general, one can find a family of collections, each containing only $k-1$ elements designated by $S_1^{k-1}, S_2^{k-1}, \cdots, S_p^{k-1}, \cdots, S_\alpha^{k-1}$. Clearly, $r_p \cup S_p^{k-1} = S^k$, $p = 1, 2, \cdots, \alpha$.

Step 2 For each S_p^{k-1} repeat step 1 to obtain a family of S^{k-2} sets. If no element can be removed from S_p^{k-1}, add S_p^{k-1} to the set CV if CV is empty. If the set $CV\{S_i^m\}$ is non-empty, add S_p^{k-1} to CV if $S_i^m \cup S_p^{k-1} \neq S_i^m$ for all S_i^m.

Step 3 Repeat step 2 until no element can be removed from any collection. Then terminate; $CV\{S_i^m\}$ is the desired result.

• *Forming relations in the third normal form*

From the definitions of normal forms, an elementary functional relation is a relation in the third form when the left hand group of attribute domains is chosen as the primary key. One might be inclined to think that the set of third normal form relations in a minimal cover could be a candidate for the canonical representation of information. Upon closer examination one finds that the number of relations is still too large. It is desirable to find a set of general rules to combine them into a smaller number of relations, yet still preserve their third normal forms. Before proceeding with the description of the combination rules, we first prove the following proposition.

Proposition 1 If $A \rightarrow B$ and $A \rightarrow C$ are elementary functional relations in a minimal cover, the composite functional relation $A \rightarrow B, C$ is a relation in the third form.

Proof To show that the relation $(\underline{A}, B, C)$ is in the third normal form, it is only necessary to show that both B and C are functionally dependent on A but are not functionally dependent on each other. Suppose B and C are functionally related by $B \rightarrow C$. Then $A \rightarrow C$ is derivable from $A \rightarrow B$ and $B \rightarrow C$ by transitivity. From the minimal cover algorithm, relations $A \rightarrow B$ and $A \rightarrow C$ could not both be members of the same minimal cover set. This contradicts the assumption of the proposition. Similarly if $C \rightarrow B$, the relation $A \rightarrow B$ is derivable from $A \rightarrow C$ and $C \rightarrow B$ and thus could not coexist in the same minimal cover set with $A \rightarrow C$. Because both $A \rightarrow B$ and $A \rightarrow C$ exist in a minimal cover, no functional relationship can exist between B and C.

The above proposition suggests the following simple rules to combine functional relations in a minimal cover.

1. Partition the minimal cover into disjoint subsets. Within each subset all functional relations have identical left hand side attribute domains.

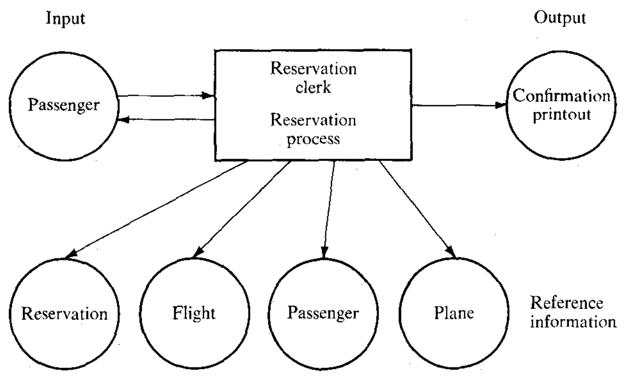

Figure 4 A simplified information flow diagram for an airline reservation system.

2. Combine each partition into a single functional relation.

The modified minimal cover set is an optimal set of relations in the *third normal form* and which has the following properties:

1. All the relations in the collection are in the third normal form.
2. The relations in the collection preserve the information about the application.
3. No other collection with fewer relations has the same set of properties.

Referring back to the example given previously with four minimal covers, we see that there is a representation for each cover. These equivalent representations are

1. $R1(\underline{E}, D, JC)$ $R2(\underline{D}, M, CT)$ $R3(\underline{M}, D)$	2. $R4(\underline{E}, M, JC)$ $R5(\underline{M}, D, CT)$ $R6(\underline{D}, M)$
3. $R1(\underline{E}, D, JC)$ $R5(\underline{M}, D, CT)$ $R6(\underline{D}, M)$	4. $R4(\underline{E}, M, JC)$ $R2(\underline{D}, M, CT)$ $R3(\underline{M}, D)$

where the primary keys are underlined.

Example of an airline reservations system

To illustrate the applicability of this synthesis technique to a realistic application problem, we consider an airline reservations system. A prospective passenger telephones an airline to indicate an intention to make a flight reservation. The reservation clerk responds to the call and obtains, through a series of questions and answers, the name of the prospective passenger, the home address and telephone number, the departure and destination cities, the desired departure date, the departure and arrival times, the number of seats, and the desired class of accommodation. The task of the reservation clerk is to verify that there is a scheduled flight between the two cities at the time and date specified by the passenger, and that the specified number of seats and class are available on that flight. If either no scheduled flight as requested can be found or no seat is available, the clerk must advise the passenger and select an alternate flight that closely matches the request. Upon receiving a verbal agreement from the passenger, the clerk must add the number of seats for the requested class of accommodation to the total number of seats already reserved; at the same time he records other pertinent information about the passenger. At this stage, the clerk is able to confirm the passenger's reservation and informs him of the pertinent flight number, date, departure and arrival airports, times, the reserved number of seats and class.

A simple information flow model for the flight reservation process is presented in Fig. 4. The process for booking reservations is partially manual because the input information from a prospective passenger is usually obtained through repeated questions and answers between the passenger and the clerk. Normally no output is generated by the process and the confirmation is conveyed verbally. When a permanent record is desirable, a confirmation slip can be printed by the system as an output. The reference information in the model consists of four major entity types, *flight*, *passenger*, *reservation* and *plane*, on which detailed information must be provided to the clerk upon request. The logical design problem is to determine, first, how each type of entity should be defined and, second, whether the four listed entity types in the model are complete to ensure the successful performance of the reservation task.

An airline reservations system expert states the factual assertions about this mini-world in the following manner:

1. A flight route is a sequence of route segments (*loc1*, *loc2*, *dt*, *at*), (*loc2*, *loc3*, *dt*, *at*),· · ·, (*locr*, *locs*, *dt*, *at*) serviced by the same airplane, where *loci* stands for location *i*, and *dt* and *at* stand for departure and arrival times, respectively. The airlines usually assign a unique flight number (*fl#*) to each flight route. For any given pair of locations *loci* and *locj*, a number of flight routes can pass through the two locations but these flight routes have different departure and arrival times.
2. When a prospective passenger makes a reservation, the following items of information have to be obtained by the reservation clerk:

Passenger Name	*psgname*	agreed to and selected by both the passenger and the clerk
Passenger Address	*addr*	
Passenger Phone No.	*phono*	
Route Segment	*loci*, *locj*, *dt*, *at*	
Date	*date*	
Flight No.	*fl#*	
No. of Seats Requested	*#seat*	
Class of Accommodation	*class*	

3. To ensure the orderly and timely sale of seats, the number of seats already reserved for each route segment must be made available to the clerk upon request:

   Route Segment
   Date
   No. of Seats Reserved, First Class *load* 1
   No. of Seats Reserved, Tourist Class *load* 2

4. The type of equipment assigned to serve a flight route determines the upper bounds of the numbers of seats available in every class.

Type of Airplane	*equipcode*
Seat Capacity, First Class	*seatcap 1*
Seat Capacity, Tourist Class	*seatcap* 2

From the above assertions, we are able to identify a number of generic terms. Each term, whether consisting of a single name or a group of names, represents an important concept. Of particular interest among term associations is the functional relation between two terms or two groups of terms. For example, a flight number *fl#* is uniquely assigned to a flight route which may contain several route segments. Thus, *route segment* → *fl#* is a functional relation between two simple terms, but *route segment, date, equipcode* → *load* is a functional relation between a group of terms and a single term. After a careful and thorough examination of the assertions previously made, one can find the following functional relations:

$$\begin{array}{l} \textit{psgname} \rightarrow \textit{addr} \\ \textit{psgname} \rightarrow \textit{phono} \\ \textit{route segment} \rightarrow \textit{fl\#} \\ \textit{fl\#} \rightarrow \textit{equipcode} \\ \textit{equipcode} \rightarrow \textit{seatcap 1} \\ \textit{equipcode} \rightarrow \textit{seatcap 2} \\ \textit{route segment, date} \rightarrow \textit{load1} \\ \textit{route segment, date} \rightarrow \textit{load2} \\ \textit{psgname, route segment, date} \rightarrow \textit{class} \\ \textit{psgname, route segment, date} \rightarrow \textit{\#seats} \end{array} \tag{1}$$

In addition to the functional relations listed above, other functional relations can be derived by means of transitivity.

It is not difficult to show that the set of functional relations listed in (1) constitutes the minimal cover and is the only cover for this application. After appropriate combinations, one obtains the optimal set of relations in third normal form:

$\textit{passenger}\ (\underline{\textit{psgname}}, \textit{addr, phono})$
$\textit{fl-assign}\ (\underline{\textit{route segment}}, \textit{fl\#})$
$\textit{eq-assign}\ (\underline{\textit{fl\#}}, \textit{equipcode})$
$\textit{plane}\ (\underline{\textit{equipcode}}, \textit{seatcap1, seatcap2})$
$\textit{open-flight}\ (\underline{\textit{route segment, date}}, \textit{load1, load2})$
$\textit{reservation}\ (\underline{\textit{psgname, route segment, date}}, \textit{\#seats, class})$

Here the names have been chosen to correspond as closely as possible to the meaning of each relation. It should be noted that in the initial design (Fig. 4) only four of the six relations are envisioned. The procedure proposed here not only leads to the precise definition of these relations but also uncovers two more relations that are an essential part of the reference information in the flow model.

Some performance-oriented considerations in defining logical segments

The rationale for defining a logical data base in terms of a set of optimal relations in third normal form has been stated succinctly by Codd in [5]. In practice, there are some cases in which further combination or decomposition of an optimal collection may be warranted because of performance-oriented considerations.

• *Access reduction in an optimal collection of relations*
In many applications, the completion of a complex transaction may involve accessing and manipulation of a number of simple facts. There are different types of facts. In an optimal collection, there tends to be a corresponding relation for each type. To minimize the response time of the system for a complex transaction may require the joining of a number of simple relations. This requirement prevails in the airline reservation system where, to process a reservation for a passenger, an instance must be inserted into the *passenger* relation while one or more instances must be inserted into the *reservation* relation. At the same time, appropriate instances of *open-flight*, *fl-assign*, *eq-assign* must be checked to ascertain that seats are still available on the requested route segment for a specific date. A feasible design tradeoff to reduce response time at the expense of some redundancy is to join the above relation in the third normal form into fewer relations in the first normal form.

• *Insertion/deletion and modification in an optimal collection of relations*
Consider two relations

$$R1\ (\underline{K_1, K_2, \cdots, K_n}, A_1, A_2, \cdots, A_a)$$

$$R2\ (\underline{K_i, K_j, \cdots, K_m}, B_1, B_2, \cdots, B_b),$$

both in the third normal form. In *R1* there are *n* domains in the key and *a* non-key domains. In *R2*, the corresponding numbers are *m* and *b*. Assume that the key domain of *R2* is a subset of that of *R1* $K_1, K_2, \cdots, K_m \subset K_i, K_j, \cdots, K_n$. After the two relations are joined on the common key domains, the resulting relation becomes $R12(\underline{K_1, K_2, \cdots, K_n}, A_1, \cdots, A_a, B_1, \cdots, B_b)$. In recording a complex fact, for every μ_1 instances inserted into relation *R1*, there are in general μ_2 instances inserted into *R2*. Clearly, to insure the state of the data base admissible with respect to the schema *R12* after each insertion, the condition $\mu_1 \geq \mu_2$ must be satisfied. In other words, no component of the primary key of any instance of the relation *R21* may have an undefined value.

The number of attribute value entries inserted for adding μ_1 instances to *R1* and μ_2 instances to *R2* is $\mu_1(n+a) + \mu_2(m+b)$, and the number of attribute value entries inserted for the same situation in *R12* is $\mu_1(n+a+b)$. Thus, the combined relation *R12* reduces the number of attribute value entries inserted if

$$\mu_1(n+a) + \mu_2(m+b) > \mu_1(n+a+b),$$

or, equivalently,

$$\frac{m}{b} > \left(\frac{\mu_1}{\mu_2} - 1\right).$$

The condition can easily be extended to three or more relations.

When a modification involves non-key attribute domains, a minimal number of instances will be updated if the collection is optimal. On the other hand, when a modification is to be performed on key attributes, say K_i, K_j, K_1, then μ_1 instances are to be updated in *R1* and μ_2 instances are to be updated in *R2*, giving rise to a total of $\mu_1 + \mu_2$ instances. For *R12*, a non-optimal relation, only μ_1 instances need to be updated.

• *Space reduction via transitive decomposition*
If the attribute domain B in a functional relation $A \twoheadrightarrow B$ has relatively few distinct values and each value occupies a great deal of space, then it may be beneficial to decompose $A \twoheadrightarrow B$ via transitivity into two functional relations $A \twoheadrightarrow B'$ and $B' \twoheadrightarrow B$, where B' is a dummy attribute introduced to establish transitivity but which has a much smaller space occupancy than B. Let p be the number of distinct values in domain A; q, the number of distinct values in domain B; l_A, the number of characters to represent each value of A; l_B the number of characters to represent each value of B; l_B' the number of characters to represent each value of B'. It can be shown that space reduction is achievable if

$$\frac{l_B}{l_B'} > \frac{p+q}{p-q}.$$

• *Logical segments for the airline reservations system*
We now make use of these practical considerations to guide the formation of segments. It is known that an airline reservation is a real time application with a very stringent response time requirement under heavy traffic. We assume that the average number of passengers making reservations in a single day is 45,000, involving seats on 50,000 different route segments. The foremost consideration in the design is to minimize the response time even at the expense of storing redundant information that is rarely updated.

An effective way to reduce the response time at this logical level is to reduce the number of relations in the optimal collection by combining *reservation*, *passenger* and *fl-assign* into a single segment, *passenger-reservation*, and by combining *openflight*, *eq-assign* and *fl-assign* into another segment, *flight*. This eliminates the need for sequentially accessing several different relations at the price of some information redundancy. The consideration of insertion activities also favors the joining of the relations *reservation* and *passenger*. In this case $n = 7$, $m = 1$, $a = 2$, $b = 2$, $\mu_1 = 50{,}000$ and $\mu_2 = 45{,}000$. Clearly, the condition $m/b > (\mu_1/\mu_2 - 1)$ is satisfied. The resulting logical segments for the airline reservation system are

passenger-reservation (*psgname, route segment, date, fl#, addr, phono, #seats, class*)
flight (*route segment, date, fl#, equipcode, load1, load2*)
plane (*equipcode, seatcap1, seatcap2*)

Summary
Traditionally, logical segments in a data base are defined on an ad hoc basis, intermixing logical representations and other performance-oriented considerations. In this paper, we have presented an approach that makes use of the functional relation to represent relevant concepts in an application. After removal of redundant relations and reduction of the number of relations, one can derive an optimal set of relations in the third normal form from the original set of functional relations. It is easier to identify primitive functional relations from the description of an application than to define relations directly; therefore, this approach is more efficient in defining the data base completely and accurately.

Performance-oriented design considerations, such as short response time, ease of maintenance, and attribute value redundancy, can be introduced separately if necessary. Specified criteria have been provided to guide the combination of relations in the third normal form into logical segments for any practical data base design.

Acknowledgment
The authors express their thanks to E. F. Codd for helpful discussions of the problem.

References
1. J. K. Lyon, *Data Base Design*, John Wiley and Sons, Inc., New York, 1971.
2. E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," *Commun. ACM* **13**, 377 (1970).
3. *CODASYL Data Base Task Group Report*, Association for Computing Machinery, New York, April 1971.

4. M. E. Senko, E. B. Altman, M. M. Astrahan, and P. L. Fehder, "Data Structure and Accessing in Data-base Systems," *IBM Syst. J.* **12,** 30 (1973).
5. E. F. Codd, "Further Normalization of the Data Base Relational Model," *Data Base Systems*, Volume 16, Courant Computer Science Symposia Series, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1972.
6. C. P. Wang, "Parameterization of Information System Applications," *Research Report RJ-1199*, IBM Research Laboratory San Jose, California, 1973.
7. T. Winograd, *Understanding Natural Language*, Academic Press, Inc., New York, 1972, p. 117.
8. W. A. Woods, R. M. Kaplan, and B. Nash-Webber, *The Lunar Sciences Natural Language Information System: Final Report*, Bolt Beranek and Newman, Inc., Cambridge, MA, 1972.
9. K. Biss, R. Chien, F. Stahl, and S. Weissman, "Semantic Modeling for Deductive Question-answering," *Proceedings of the Third Joint Conference on Artificial Intelligence*, Stanford University, Stanford, CA, 1973, p. 356.
10. "Application Customizer Service, System/3, Application Description Manual," *Manual GH20-0628*, IBM Corporation, White Plains, NY, 1966.
11. W. A. Martin and R. Krumland, "MAPL, A Language for Describing Models of the World," *Automatic Programming Internal Memo 6*, Project MAC, MIT, Cambridge, MA, 1972.
12. M. M. Hammer, W. G. Howe, and I. Wladawsky, "An Interactive Business Definition System," *ACM-SIGPLAN Very High Level Language Symposium*, Santa Monica, CA, March 1974.
13. The Language Structure Group of the CODASYL Development Committee, "An Information Algebra," *Commun. ACM* **5,** 190 (1962).
14. C. Delobel and R. G. Casey, "Decomposition of a Data Base and the Theory of Boolean Switching Functions," *IBM J. Res. Develop.* **17,** 374 (1973).
15. A. V. Aho, M. R. Garey, and J. D. Ullman, "The Transitive Reduction of a Directed Graph," *SIAM J. Comp.* **1,** 131 (1972).
16. C. Delobel, "A Theory About Data in an Information System," *Research Report RJ-964*, IBM Research Laboratory, San Jose, California, 1971.

Received March 5, 1974; revised September 6, 1974

Dr. Wang is located at the Advanced Systems Development Laboratory, Yorktown Heights, New York, 10598; Dr. Wedekind is at the Technical University of Darmstadt, 61 Darmstadt, West Germany.