This paper describes the design effort for an integrated data base and then develops techniques for automating significant portions of the labor. These techniques have been incorporated in a program to provide an effective data base design tool (Data Base Design Aid) in current use. The processes involved with this aid are discussed.

Automated logical data base design: Concepts and applications

by N. Raver and G. U. Hubbard

Designing an integrated data base is currently a costly and time-consuming activity. An integrated data base exists to serve many application functions and its design must consider the requirements of the functions it will serve. When inconsistencies and redundancies exist in these requirements, design trade-offs must be made. Once the data requirements have been identified and recorded for all the application functions that will use the data base, the designer must subject these requirements to examinations and analyses which can be tedious and time-consuming. The goal is to devise a data base having a controlled number of redundant data elements connected only by those relationships required to support each application function with reasonable performance. The data base should also be extendable, without restructuring, to new functions. Automatic computations can assist in this analysis, helping the designer produce a better data base in a shorter period of time.

Data requirements are determined by listing all data elements required by each application function and by defining how these elements are related. The designer usually begins by examining the specifications of the functions that will use the data base. After collecting the names of the required data elements, the designer studies a sorted list of these names to identify and resolve inconsistent usage. Frequently the same name (e.g., DATE) will have been used many times to mean different things (homonyms). Also, where standardization is not sufficiently developed, different names (e.g., EMPLOYEE and MAN-NO) may have been used to mean the same thing (synonyms).

Next, the relationships defined on the data elements are examined, and the designer determines which elements will serve as keys (sequence fields) and which will be attributes (nonkeys). Then, depending on the rules of the data base handler to be used, the designer organizes the data elements into storage patterns for implementation. For the Information Management System (IMS/VS), the designer will group the elements into segments, group the segments into hierarchical trees, and determine the locations and the nature of logical relationships and secondary indexing.

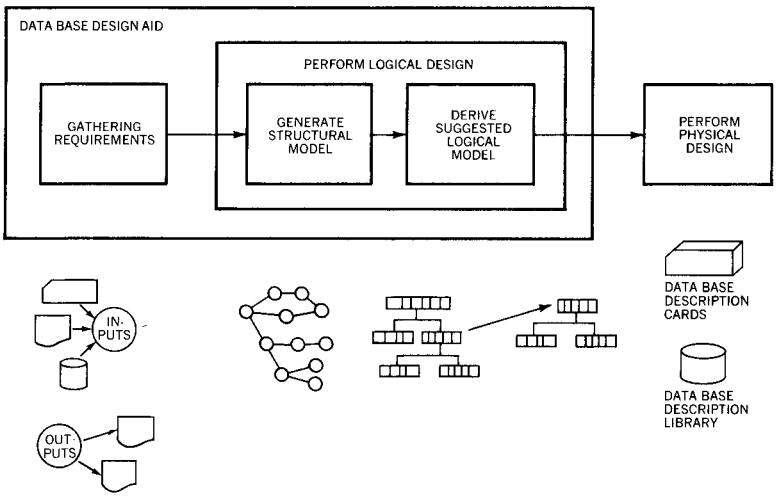
The problems of data base design are twofold: (1) it can be a lengthy and time-consuming process; and (2) the desired quality is elusive. Automated methods are available, however, to provide assistance in both areas. The designer using manual analysis methods can be confused by the mass of detail, and mistakes and rework often result. In an actual DL/I (Data Language/I[1,2]) design study using these automated techniques, there were 1818 relationships defined on 1588 elements from which 105000 possible hierarchical paths were deduced. The procedure is quite simple, but manual analysis becomes unwieldy when the number of elements and relationships is large.

By automating certain parts of the logical design process, the designer is relieved of much drudgery and is able to produce a better final design. These automated techniques do not remove the human from the design process; they merely reduce much of the work of performing the routine and tedious tasks.

Quality in a data base design depends largely on the expertise of the human designer. Automated methods can assist the designer by identifying problems and design alternatives. They can help in identifying homonyms and synonyms, and they can detect many types of inconsistencies and incompletely defined requirements. In addition, nonessential (alternate) paths defined between the same pair of elements can be identified for possible exclusion from the design. In all these areas, the designer is provided with information that is needed and which is likely to be more complete than if derived manually. Because the computations follow well-defined algorithms, the logical design suggested by these procedures may offer new perspectives or insights to the designer. Initial experience with these techniques has shown instances where the designer's viewpoints have been clarified or altered by the computed results.

An important by-product of using automated methods is the rigor imposed on data gathering. The designer must analyze the requirements of the application functions thoroughly and record

Figure 1 Data base design effort



the requirements with completeness in a precise and consistent format. A significant reduction of errors, omissions, and inconsistencies can be expected.

Automated logical data base design is addressed first as computational concepts with their associated algorithms, procedures, and techniques. Following this discussion, a description of an IBM program product, Data Base Design Aid[3] (DBDA), is presented. DBDA applies the concepts of automated logical data base design to produce logical designs for DL/I structures. This discussion also outlines how a designer would use DBDA reports to obtain the final logical design.

The concept of automated logical data base design

overall data base design process

The overall data base design effort, illustrated in Figure 1, can generally be divided into the following three steps:

1. *Gathering and Recording Data Requirements*—Gathering and recording data requirements involves analyzing the application functions that will use the data base and determining the data requirements of each function. The requirements may be categorized as output, input, or processing. For each function, the designer identifies and records the names of the data elements required in the data base and the type of association defined on each pair of related data elements.
2. *Deriving a Logical Design*—Logical design involves organizing the collected data requirements into a structural model, which is a nonredundant network of the data elements and

their relationships, and then deducing a logical model of keys, attributes, segments, hierarchical paths, logical relations, and secondary indexing.

3. *Constructing a Physical Design*—The logical design can be physically implemented in a variety of ways in order to achieve the normal access method trade-offs. These trade-offs involve performance parameters such as retrieval volume, update volume, packing density, periodic reorganization, load and processing time, etc. In a system such as IMS/VS, this would involve the selection of options for a data base design such as device type, access methods, pointer options and data set groups.

Once the data requirements are known and recorded, automated procedures can be used to analyze them for duplications, inconsistencies, omissions, and alternatives. After the designer resolves these issues, further computations may produce a suggested logical design. Some of the information derived in this process can also be helpful to the designer when constructing his physical design.

Since data base nomenclature and terminology is frequently overlapping and conflicting, basic terminology requires definition. The following are defined aspects of data elements and associations.

data elements

A *data element* is the smallest nondivisible data reference permitted. A data element is the symbolic reference to all occurrences of that data element, and a *data element occurrence* is a specific value.

There are two kinds of data elements: *keys* and *attributes*. A *key* is a data element whose occurrences are unique and whose values are used to identify corresponding values of a related data element. An *attribute* has values which are not necessarily unique.

A *segment* is a group of data elements. Each segment occurrence is uniquely identified by a key or by its relative position.

A key that consists of more than one data element is called a *compound key*. If A, B, and C form a compound key, then write (A*B*C), where the * is the concatenation operator. A *full compound key* is one in which every data element is a valid key in its own domain, such as (STUDENT*CLASS). A *qualified compound key* is one containing a member data element that is not a unique identifier, such as (SALES-ORDER*LINE-ITEM-NO). The line item numbers are needed but are not unique; however, each line item needs the qualified compound key for unique identity.

A *synonym* refers to two or more data element names used for the same data. Synonyms should be detected and a consistent terminology adopted. Automated techniques can provide assistance in detecting synonyms.

A *homonym* refers to a data element name that actually means two or more different things. Homonyms must be identified and resolved. Automated techniques can significantly aid in detecting them.

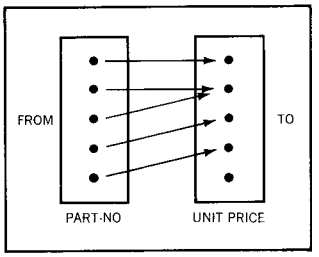
associations

An *association* is a from-to relationship between two data elements A and B, and is specified as (A,B). Note that an association is in one direction only. The "from" element serves as an identifier, and each of its occurrences identifies one or more (or no) occurrences of the "to" element.

Three basic types of associations will be considered: the simple association, the complex association, and the conditional association.

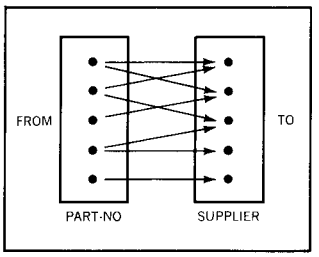
A *simple association* (*Type 1*) is one in which every occurrence of the "from" element identifies one and only only occurrence of the "to" element. The association (PART-NUMBER, UNIT-PRICE) is a simple association when a given part has only one price. Note that several "from" occurrences may identify the same "to" occurrence; that is, several parts may have the same unit price, but each "from" occurrence always identifies one and only one "to" occurrence. Figure 2 shows a simple association.

Figure 2 Simple association



A *complex association* (*Type M*) is one in which each occurrence of the "from" element can identify any number (including zero) of occurrences of the "to" element. An example of a complex association is (PART-NO, SUPPLIER)—a given part number can be furnished by several suppliers. Thus, in a complex association, a given occurrence of the "from" element does not uniquely identify an occurrence of the "to" element but, in general, identifies *many* "to" occurrences. Figure 3 shows a complex association.

Figure 3 Complex association



A *conditional association* (*Type C*), depicted in Figure 4, is a special case of both the simple and complex associations. Each occurrence of the "from" element identifies either one or no occurrence of the "to" element; i.e., the "to" occurrence may or may not exist. A conditional association is illustrated by (EMPLOYEE, SPOUSE)—a married employee will have a spouse and a single employee will not. Thus, under some conditions a "from" occurrence will have a corresponding "to" occurrence, and under other conditions it will not. When the "to" occurrence exists, there is one and only one value associated with a "from" occurrence.

Figure 4 Conditional association

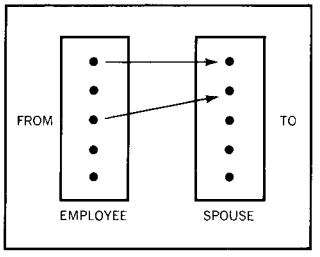


Figure 5 Multiple-meaning association

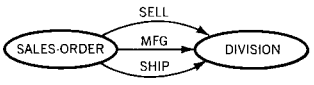
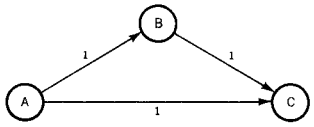


Table 1 Six mapping cases

	Forward	*Backward*
Mapping 1	M	1
Mapping 2	C	1
Mapping 3	1	1
Mapping 4	M	M
Mapping 5	M	C
Mapping 6	C	C

Figure 6 Type 1 association

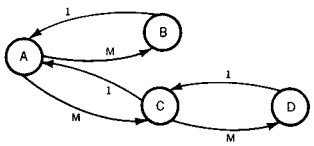


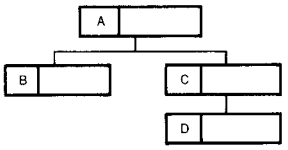
When there is more than one relationship between a pair of data elements, we call this a *multiple-meaning association*. For a more mathematically oriented description of this concept, see References 4 and 5. If multiple-meaning associations are defined between A and B, then a label is used to distinguish the difference. It is denoted as (A,B):LABEL. Thus, in Figure 5 we have

(SALES-ORDER,DIVISION):SELL
(SALES-ORDER,DIVISION):MFG
(SALES-ORDER,DIVISION):SHIP

which designate, respectively, the company divisions that sell, manufacture, and ship items on a given sales order.

Figure 7 Parent-child relationships





An *essential association* is a Type 1 association that provides the only path between two data elements. An *implied association* is a Type 1 association defined between two data elements between which a path of essential associations already exists. As shown in Figure 6, (A,B) and (B,C) are essential associations, and (A,C) is an implied association.

For every forward association, an inverse association can be defined by reversing the "from" and "to" roles of the data elements. However, if the forward association is simple (Type 1), the inverse association may be complex (Type M). There is no real ordering; therefore, either association can be called forward and the other becomes the inverse. An association and its inverse is called a *mapping* and is written as (X:Y), where X is the type of the forward and Y the type of the inverse association. Because mappings are symmetric, there are six cases to consider as listed in Table 1. This discussion applies only to mapping between keys.

(M:1) mapping

The (M:1) mapping defines a parent-child relationship between the pair of related keys. A parent may have many children (i.e., many occurrences of a child segment type); a child can have only one occurrence of its physical parent. A series of keys connected by (M:1) mappings may represent a multilevel hierarchical structure as shown in Figure 7.

(C:1) mapping

The Type C association between keys can be considered a special case of the Type M association. A parent may or may not have a child, but if the child exists, it has only one physical parent; therefore, a parent-child relationship is defined in which the child segment may or may not exist.

(1:1) mapping

The (1:1) mapping is an identity in which an occurrence of each element uniquely identifies an occurrence of the other. Although a parent-child relationship can be implied (a parent segment may have only a single occurrence of a child segment type), the designer usually prefers to implement an identity by either discarding one of the elements or by making one element an attribute in the segment keyed by the other element. A secondary index may be desired to access the element that has become the attribute.

(M:M) mapping

The (M:M) mapping defines candidates for logical relationships as depicted in Figure 8. A given occurrence of one data element may define many occurrences of the other, and vice versa.

(M:C) mapping

(M:C) mapping indicates that while a parent may have many children, a given child may or may not have a parent. Such structures cannot occur in DL/I. This mapping is treated as an (M:M) mapping because it can be implemented only in a logical relationship.

(C:C) mapping

Element A may or may not have a B. If element B exists, it may or may not have an A. This (C:C) mapping is also treated as (M:M) for implementation through a logical relationship.

Three levels (or views) of data are associated with an integrated data base and its application functions:

1. *External View*—The external view is visible data as presented on output reports, displays, etc., and on input sources, and it therefore corresponds to the structure of data as it appears to a user at a terminal or a user reading a report printed by the system. The external views are normally defined in the functional specifications of the applications.
2. *Local View*—This view represents that portion of the integrated data base required to support a particular application function to generate the external view or, in the case of update, to absorb the external view. The collection of local views for the various application functions using the data base determines the requirements of the external view.
3. *Internal View*—This view is the complete data structure maintained by the system to generate the multiple local views. The internal view (sometimes called the associative model) describes the integrated data base itself.

Figure 8 (M:M) mapping

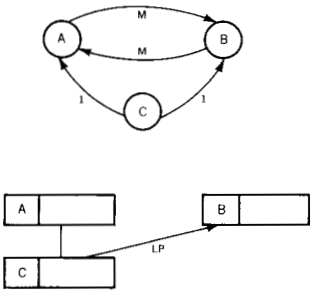
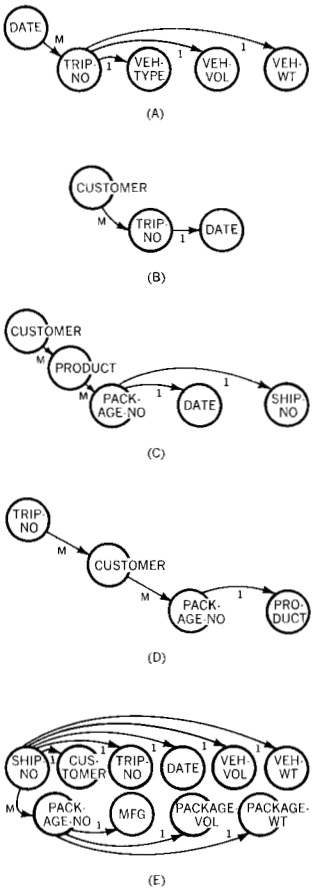


Figure 9 Five application functions



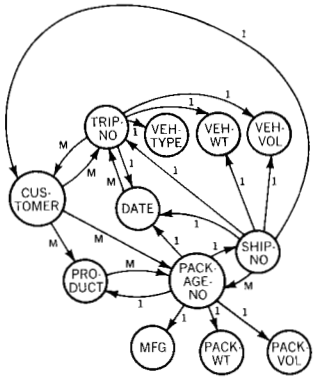
Starting with the external views and considering special processing requirements, the designer and application specialist determine the required local view for each application function. This process is data gathering and recording. Once the local views are obtained, automated processing can help in deriving the required content and structure of the internal view, which is the end result of logical design. (The local view is analogous to the application function's PSB (Program Specification Block); the internal view is analogous to the data base's DBD (data base description).)

An example of automated data base design

The concept of automated data base design is basically very simple and involves using the computer to analyze and process the data base requirements as an aid to the human designer. It is an iterative process in which the designer (1) defines the data requirements of each application function to use the data base, (2) uses the computer to combine these requirements into a structural model, analyze them, and derive a representation of a logical design, and (3) tests the resulting representation against each application's requirements to see if its functional and performance requirements are supported. When necessary, the data requirements may be modified and the process repeated until a suitable representation is obtained. This final representation is the desired logical design of the data base.

This procedure is illustrated by an example in which a simple data base for a trucking company is designed. (Even in this simple example, the final result is not initially obvious, and the reader is cautioned not to read ahead so that the complexity of small networks and the simplicity of the procedures can be appreciated.) This example illustrates only the basic automated methods. Editing functions such as resolving synonyms and homonyms are not illustrated.

Figure 10 Composite network



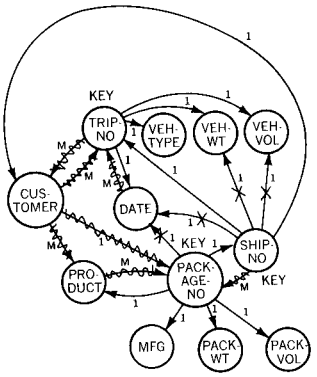
A data base is being designed for a trucking company that loads its trucks with products for shipment to various customers. Many trips are made each working day, and each trip is made by a certain type of vehicle. Each component of a product is given a package number. On a specific trip, all packages for a given customer are grouped and given a single shipment number.

The data base is required to support five application functions that provide operating information for the company. A schematic representation of each function is depicted in each part of Figure 9. (For simplicity, only the output requirements of each function are considered.)

Part A of Figure 9 shows the trip schedules (Local View 1) that list each trip by date, and for each trip, give the vehicle type, weight, and volume. The customer shipment query (Local View 2) shown in Part B handles customer queries about the dates of scheduled trips to a customer. Part C illustrates the customer product query (Local View 3) that handles customer queries such as, "When and what is the shipping information for given products?" The trip contents (Local View 4) lists each trip, the customers to be served, and the packages and products to be delivered as shown in Part D. The shipment history (Local View 5) in Part E provides a history of each shipment.

A canonical representation of the logical design is derived by combining the five local views into a single, composite network (structural model) of data elements and associations and analyzing this network. A glance at the network in Figure 10 shows a confusing picture at first. Which elements are keys and which are attributes? Can hierarchical trees (assuming a DL/I system) be deduced? Which are the root keys? Are logical relationships required?

Figure 11 Network essential associations



canonical representation

The analysis will identify and eliminate nonessential associations, determine keys and attributes, and derive a logical model according to the rules of the data base handler to be used. At the same time the procedure should reveal errors, omissions, and inconsistencies in the data requirements and conditions that violate some of the data base handler rules. In addition, design alternatives are provided for resolution.

Physical hierarchies and segment content are deduced from essential associations. To focus on the essential associations, the complex associations and the implied simple associations are marked for removal from the network in Figure 11.

After removal of those associations, keys and attributes are determined. Keys are those elements from which a Type 1 association originates, as indicated in Figure 11. Attributes are elements identified by a Type 1 association but which do not in turn identify anything else with a Type 1 association. Three keys are determined: TRIP-NO, SHIP-NO, and PACKAGE-NO. By applying these concepts to the example at hand, the segment structure and hierarchical relationships become recognizable and can be explicitly depicted, as in Figure 12.

This example illustrates the basic concepts of automated logical data base design for deriving segments and physical hierarchies. Later in this paper, it will be shown how Type M associations are analyzed to identify candidates for logical relations and for secondary indexing.

Figure 12 Segment structure and hierarchical relationships

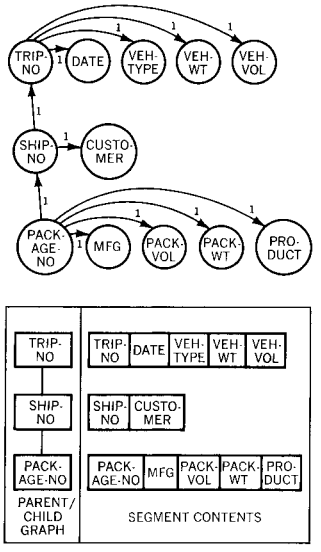
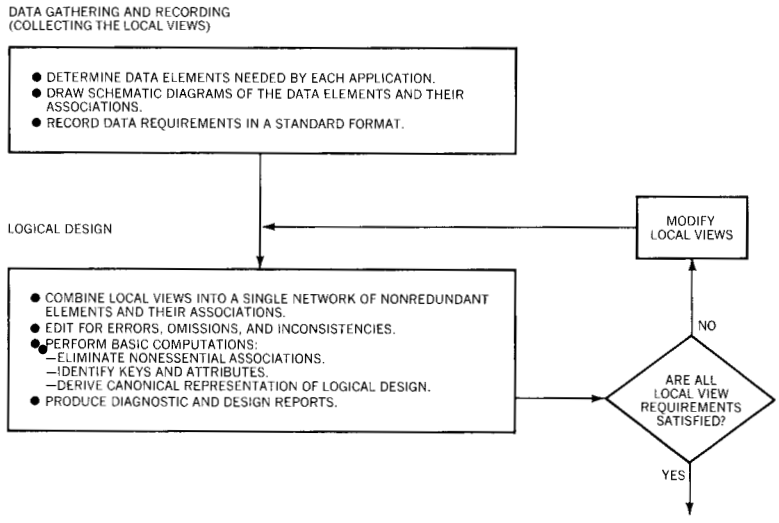


Figure 13 Automated logical data base design process



testing the canonical representation

Having obtained the canonical representation of the logical design, the designer's third task is to test that representation against each of the local views to see if it contains the required access paths and to see if reasonable performance can be expected. If yes is the answer in both cases, the logical design is complete, and the designer can proceed with the physical design. Otherwise, the designer must augment one or more local views and repeat the process. The flowchart in Figure 13 shows the process.

The original local views are termed "unrestrained." Modified local views are "restrained," since limitations are imposed upon them because of the data base handler rules, or by performance and maintenance considerations.

We illustrate these concepts by testing the canonical representation against each of the original five local views from which it was derived.

View 1 can be satisfied provided a secondary index is implemented with DATE as source and TRIP-NO as target.

View 2 requires a secondary index with CUSTOMER as source and TRIP-NO as target. Note that the entire TRIP-NO segment will be presented to application Function 2.

View 3 is not efficiently supported by the canonical representation. One and possibly two sorts will be required to produce the report. A secondary index with CUSTOMER as source and SHIP-

NO as target will avoid an additional sort. The designer may wish to reconsider and modify View 3 (producing a restrained load view) to avoid the sorting.

View 4 is directly supported by the canonical representation.

View 5 requires a secondary index on SHIP-NO and a backward pointer from the SHIP-NO segment to the TRIP-NO segment.

Programming considerations

Automated data base design depends on the data model supported by the data base handler. DL/I (IMS/VS or DOS DL/I) has the following rules and restrictions. (This list is intended to be representative and not exhaustive.)

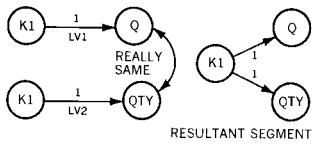
1. A segment consists of a collection of fields (data elements), one of which may be a key while the others are attributes uniquely determined by the key.
2. A segment (other than the root segment) has one physical parent and may or may not have a logical parent. No more than one of each parent is permitted.
3. A logical relation between two segments requires at least a third segment to relate their occurrences. The "intersection" segment is a physical child of one parent and a logical child of the other.
4. A logical child of a logical child is not permitted.
5. DL/I permits redundant data elements but not redundant segments.
6. Multiple-meaning associations are not permitted in DL/I. They can be implemented by creating separate "to" elements or by logical relations.
7. The number of segments in a hierarchical path may not exceed 15.
8. A segment may not be the physical parent of a superior segment of its hierarchical path; i.e., loops cannot be implemented without using at least one logical relation.
9. An association is not declared explicitly in DL/I but is implied by either the physical hierarchical paths or by logical relationships.
10. Secondary indexes may allow any key or attribute to be used as a point of entry to the structure.

automated analysis

The following are some of the diagnostics possible in automated logical data base design:

1. Suppose we have two local views, all dealing with a key K1 and a single attribute, Quantity. Suppose further that the first view calls the attribute QTY, and the second calls it Q. The

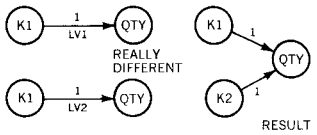
Figure 14 Two views with same attribute



basic computation will create a segment with K1 and containing QTY and Q, shown in Figure 14.

Because all three data item names appear together in the same segment, the system designer will have little difficulty in noting inconsistencies within a single segment. A report of each segment can be produced which lists each attribute in close proximity. While more elaborate schemes can be devised to handle synonym problems, they involve more labor in collecting local views and are unwieldy in practice.

Figure 15 Two views with different attributes

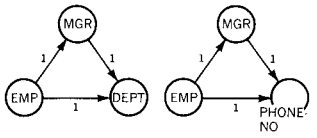


2. Now suppose we have two local views dealing with two different attributes which are erroneously called the same data item name, QTY, as shown in Figure 15. The first view uses key K1 for its attribute QTY, and the second view uses K2. In this homonym problem, a report can be produced that gives a diagnostic for QTY since it has more than one key. A number of commonly used terms such as DATE, NUMBER, and COST are likely to create a homonym problem.

3. Inconsistent associations can also be detected. If one local view contains (EMPLOYEE,DEPT) as Type M and another local view contains (EMPLOYEE,DEPT) as Type 1, then one must be wrong. For DL/I implementation, either the association must be of the same type, or two "to" elements must be used.

   A list of inconsistently defined associations (and the local views in which they were specified) can be produced. This listing may also help to identify any multiple-meaning associations.

Figure 16 Implied association



4. Implied associations (the transitivity property) can be detected and reported. Such associations will normally be considered nonessential and will be eliminated by the program. In Figure 16, aside from the obvious question of throughput, (EMP,DEPT) can be eliminated without loss of validity, whereas (EMP,PHONE-NO) must be retained to get the employee's correct phone number. Means must be provided to retain selected implied associations in the model.

5. A path of simple associations that loops back onto itself cannot be implemented as such in DL/I structures. In some cases (Bill of Materials is a simple example), this situation is valid, but it should be reported. If it is valid, implementation requires that at least one simple association be replaced with a logical relation.

6. Data element pairs related by (M:M) mappings can be implemented in parent-child structures with possible redundan-

cy or through logical relations. In the latter case, a logical child (intersection segment) is required. This situation and the presence or absence of a defined intersection can be reported for review.

7. If frequencies of use can be estimated for the associations defined in each local view, relative frequencies of use can be calculated for the resulting paths of the integrated data base (internal view). This information can assist the designer in decisions where performance is a factor.

reports

The input editing of the data requirements, the analyses indicated above, and the basic computations of automated logical data base design can be reported for review and for documentation. The following reports will be useful:

1. An edit listing of input to reveal errors, omissions, and inconsistencies that can be detected before the analysis begins.
2. Lists of data element names alphabetically sequenced and in keyword-in-context sequence containing, for each name, sources, uses, and all supplied characteristics.
3. A diagnostic list of attributes having multiple keys for essential associations.
4. A diagnostic list of associations between keys for which the mapping is incomplete; i.e., the association is known in only one direction.
5. A report of inconsistent associations.
6. A report of "implied" associations to be eliminated by the program.
7. A report of (M:M) mappings that are candidates for logical relations, that also show which intersections are already defined.
8. A report of the network showing all keys and their essential associations.
9. A report showing the relative frequencies of use of the paths in the network.
10. A list of segments with their contents.
11. A diagram of the physical structures derived from the network. In DL/I, this is the parent-child graph.
12. A list of candidates for logical relations and secondary indexing.

The methods previously discussed for automated logical data base design have been implemented in the program product Data Base Design Aid (DBDA). DBDA provides direct support for DL/I data structures, but it can also be used in designing structures for other data base handlers. It runs under both OS/VS and DOS/VS.

phases and iterations

DBDA executes as a batch program. It is organized in six phases, with one or more job steps in each. The first phase edits the data requirements, and the final phase produces a series of design reports defining the suggested logical model. The intermediate phases produce diagnostic reports that the designer may use to suspend processing, modify the data requirements of the control parameters, and restart the process at certain points.

DBDA input

There are two types of input to DBDA: the data requirements (recorded on DBDA Requirement Specification Forms), and a set of control parameters (specifying editing functions, special processing requirements, and special characters for output formatting). Coding details are presented in Reference 6.

The following information must be provided in the data requirements.

1. The names of the data elements required by each application function using the data base.
2. Designation of the identifier and the identified, for related elements.
3. Association type for each pair of related elements.

If information can also be provided regarding the frequency of use of the application functions and the frequency and nature of access to the data elements used by each function, DBDA can calculate relative measures of the importance of the paths of the resulting hierarchical structures.

DBDA can also accept information regarding the characteristics of the data elements, (e.g., data type, length, length type) for use in consistency editing.

DBDA output

The design reports of DBDA describe the suggested logical model of the data base. The edit and diagnostic reports provide the diagnostics mentioned earlier. All reports are fully described in Reference 7. Brief descriptions of the design reports follow.

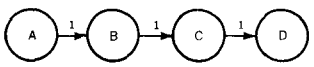
- *Parent/Child Graph*—A graphical presentation of the physical hierarchical structures derived from the structural model.
- *Suggested Segments*—A list of all keys and all attributes clustered about each key. For each suggested segment, the report shows key name, attribute names, indication of fixed or variable lengths, and segment size.
- *Structural Model*—A listing of the network of all keys and their associations, showing all possible parent-child relationships. For each key possible, alternate parents (candidates for logical relationships) are listed. A summary of the association of weights is given for each path.

- *Candidates for Secondary Indexes*—Any data element that appears as a root in the requirements of some application, but is not a root in the suggested hierarchical structure, is listed as a candidate for secondary indexing. Data elements that were entered as identities but changed to attributes are listed. Attributes that refer back to their keys or to other elements with Type M associations are also listed.
- *Association Weights*—Association weights indicate the relative frequency of use and importance of the paths defined in the structure. Results are given for batch and on-line environments, and provision is made for special weighting of insertions, replacements, and deletions. This report helps the designer make decisions that involve performance considerations.

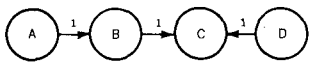
DBDA processing

building paths

A path is defined to be two or more data elements in a linear array such that adjacent element pairs are related by Type 1 associations all in the same direction. For example, this array is a path of four elements:



The following array is not a path:



However, two paths can be constructed from it.

DBDA is interested in paths of Type 1 associations because they constitute the basis of physical hierarchical structures. The (M:1), (C:1), and (1:1) (if properly implemented) mappings constitute parent-child relationships, with Type 1 associations (child to parent) being the essential ingredient. Thus DBDA builds and analyzes all possible paths of element pairs connected by Type 1 associations.

analyzing the paths

DBDA attempts to map the data elements into physical hierarchies by joining paths of "essential" Type 1 associations. The analysis begins by scanning all possible paths and categorizing their associations into one of three groupings: Essential Associations, Implied Associations, and New Associations. From the paths of Essential Associations, DBDA builds the physical hierarchies. To illustrate, suppose the following association pairs are among those specified in the data requirements:

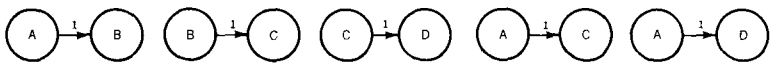
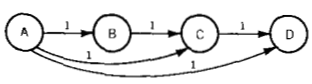


Table 2 Summary of association groupings

Essential Associations	*Implied Associations*	*New Associations*
(A, B)	(A, C)	(B, D)
(B, C)	(A, D)	
(C, D)		

Figure 17 Network of paths



From these pairs, DBDA deduces the network of paths in Figure 17. (A,B), (B,C), and (C,D) are the essential associations. (A,C) and (A,D) are implied associations because their functional capabilities are implied by the essential associations. (B,D), although not specifically requested, is intrinsic to the structure as a "new" association. "New" associations indicate queries that can be made against the data base. A list of "new" associations is also useful for judging if the data base already has the necessary associations for supporting new functions. Table 2 summarizes the association groupings of this example.

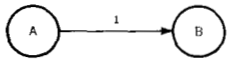
deducing keys, attributes, and segments

Data elements that identify other data elements with simple associations are classified as keys. Data elements that are not keys and that are identified by simple associations are classified as attributes. A key and the attributes it identifies are grouped into suggested segments. (In DBDA, the name of the key is also used as the segment name.)

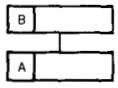
Frequently a designer will specify only one association (called the forward association) between a pair of data elements, so that DBDA must assume the inverse association. These assumptions affect the determination of keys, attributes, and segments. In the following discussion, A in all cases is considered to be a key.

Case 1.

(A,B) = 1 B is a key.

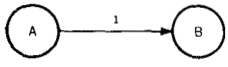


The inverse association is assumed to be Type M, and since B (as a key) represents a segment, A and B are mapped into a parent-child relationship.



Case 2.

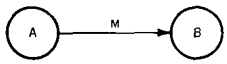
(A,B) = 1 B is not a key.



The inverse association is not needed. B is treated as an attribute and becomes a field in the segment keyed by A.

Case 3.

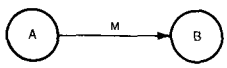
(A,B) = M B is a key.



If the inverse association is assumed to be Type M, the A and B segments become candidates for a logical relationship, and DBDA looks to see if an intersection segment is defined. If the inverse is assumed to be Type 1, DBDA creates a parent-child relationship. The designer may instruct DBDA to assume either inverse.

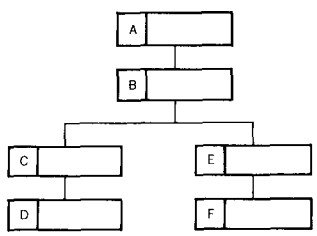
Case 4.

(A,B) = M B is not a key.



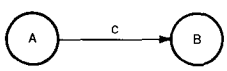
B is considered to be a "multiply occurring" attribute (i.e, a given occurrence of A identifies many Bs). More information is required for implementing this association. The designer has two choices: (1) Implement B as a COBOL repeating group within A. This may be specified by defining a Type 1 association from A to B and by defining B as a variable-length field. (2) Implement B as a dependent segment of A. This may be specified by explicitly defining a Type 1 association from B to A.

Figure 18 Hierarchical tree



Case 5.

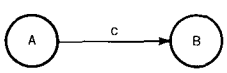
(A,B) = C B is a key.



DBDA treats this association type as Type M. Thus, this case is treated as in Case 3.

Case 6.

(A,B) = C B is not a key.



The Type C association is treated as a Type 1, and the inverse association is not relevant. B becomes an attribute (which may or may not have a value) in the segment keyed by A.

deducing hierarchies

Hierarchical tree structures are built by combining paths. As an example, the hierarchical tree in Figure 18 is formed by combining the following paths, in which all elements are assumed to be keys:

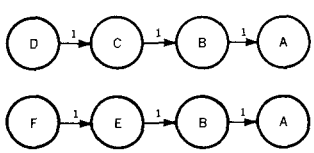
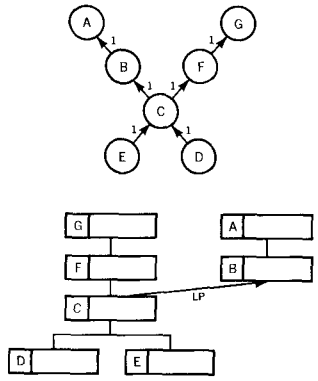
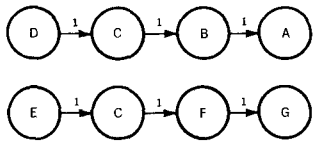


Figure 19 Hierarchies resulting from selection of F as parent



However, for the following two paths, the resulting structure has two parents, B and F, of C:



In DL/I hierarchies, there can be only one physical parent; therefore, a choice must be made between B and F as the physical parent of C. If F is selected, the resulting hierarchies are shown in Figure 19.

Thus, in producing the Parent/Child Graph, the segments having more than one parent are identified and a dominant (physical) parent selected. The designer may make these choices, or DBDA can make the selection by choosing the parent on the path having the highest association weight between candidate parent and child, or, if the associations' weights are absent or equal, by selecting the parent whose name is first in the collating sequence.

calculating association weights

Association weights are a relative measure of the frequency of use (or importance) of the paths in the structural model. Such a measure, if reasonably well-estimated, provides the designer with valuable insight into performance-related questions such as:

- Which segments to place into the left-hand and into the right-hand paths.
- Whether to subdivide segments into frequently used fields and occasionally used fields.
- Whether to implement certain pairs of elements in a logical relationship or in a parent-child relationship with possible inverted searching.

The calculation of the association weights is described in Reference 5.

analyzing candidates for logical relations

As has been stated, the (M:M) and (M:C) mappings are candidates for logical relations. The incomplete Type M mapping from A to B (where B is a key) may also be considered as a candidate. For each candidate pair of elements, DBDA searches for paths that will define the intersection for DL/I implementation. It searches first for paths defining a common child. The intersection may be defined by a single data element as in Figure 20A, or it may be defined by multiple data elements, as in Figure 20B.

DBDA restricts this search to a single logical crossing between physical structures; therefore, situations like that in Figure 21

are not analyzed. If a common child is not found, a search is made for a common parent as shown in Figure 22.

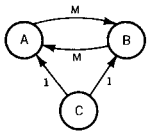
A common parent may or may not define the intersection, depending on the number of occurrences of B or C. It is presented to the designer merely for evaluation. If neither a common child nor a common parent is found, the logical relation cannot be implemented without further definition.

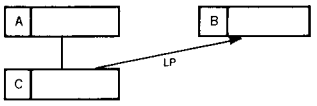
determining candidates for secondary indexing

A data element that appears as a root key in the requirements of some application function, but does not become a key in the root segment of the suggested logical model, is a candidate for secondary indexing. As an example, an application function requires data elements A, B, and C, and another application function requires data elements B, D, and E. For the resulting hierarchical structure, element B is listed as a candidate for secondary indexing as shown in Figure 23.

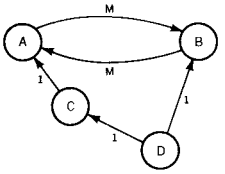
In addition, if an identity (i.e., two elements related by a (1:1) mapping) is resolved by making one element an attribute of the other (see DBDA's IDENTITY command in References 6 and 7), the new attribute is listed as a candidate for secondary indexing as in Figure 24. Any attribute which refers back to its key or to other elements with a Type M association is also listed.

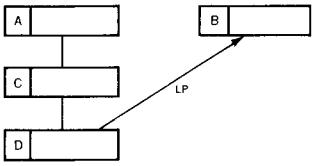
Figure 20 Data element intersection





(A)





(B)

Figure 21 Nonanalyzed situation

Using the results of DBDA in logical design

While DBDA produces design reports showing a suggested logical design of the desired data base, the designer still must make design decisions, some of which may involve reorganizing segments or rearranging suggested structures. This section indicates how the designer proceeds from the DBDA design reports to the final logical design. The designer controls the process, and DBDA provides helpful information.

The example used is the trucking company described earlier in this paper. In that exercise, a single data base of three segments was derived, and CUSTOMER, a root key of two local views, was not a root of the resulting structure. It was not even a key because no attributes had been defined for it.

Assume that one or two additional DBDA iterations have been performed in which CUSTOMER and PROJECT now have attributes and are keys of segments, and all diagnostics are resolved. The resulting five design reports are Figures 25 through 29. Because DL/I does not permit a logical child of a logical child, the suggested logical design must be refined. This can be done in several ways, and the choice belongs to the human designer. The solution chosen for this iteration may not be the best solu-

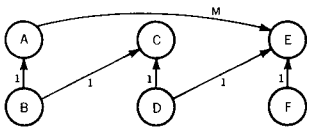


Figure 22 Search for common parent

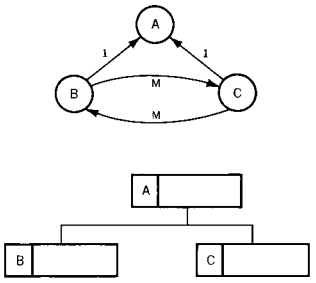


Figure 23 Secondary indexing structure

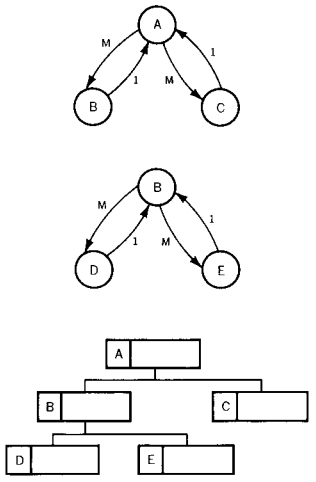
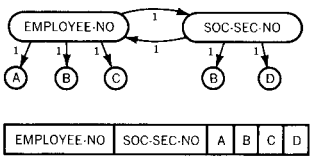


Figure 24 New attribute in secondary indexing



tion. The purpose of this example is (1) to show the value of the reports of DBDA in helping the designer choose a solution and (2) to indicate the value of DBDA in documenting the refined design and in reporting any further anomalies or alternatives created by the refinement.

Clues for the refinement to be made are obtained by studying the associations and the association weights for the paths involved in the two sets of candidates for logical relations (Figure 30). The following observations are pertinent.

1. The highest association weight, 41280, is for paths from PRODUCT to PACKAGE-NO and from PACKAGE-NO to SHIP-NO. These paths require the greatest accessing efficiency.
2. The Suggested Segments report shows that SHIP-NO is in a segment without any attributes. Perhaps SHIP-NO can be moved into the PACKAGE-NO segment.
3. The lowest association weight, 1, is from SHIP-NO to each of its two parents, CUSTOMER and TRIP-NO, and there is no requirement to go from either of these parents down to SHIP-NO. A rearrangement will have relatively little impact on performance.
4. A complex mapping is defined between CUSTOMER and TRIP-NO, and these two keys are strongly suggested to be parents in a logical relationship. If SHIP-NO is moved, a new intersection must be defined.

The chosen solution, then, is as follows: The SHIP-NO segment is to be removed from the structure, and SHIP-NO will be placed as an attribute in the PACKAGE-NO segment. SHIP-NO will be needed as the source of a secondary index to CUSTOMER, and this should be reflected in the Candidates for Secondary Indexes report. A new pointer segment will be created to define the intersection between CUSTOMER and TRIP-NO.

Methods for communicating these refinements to DBDA are described in Reference 6. An additional DBDA iteration will document these refinements and determine whether additional errors and alternatives are created. In this case, the desired design is produced without diagnostics requiring remedy. The new Parent/Child Graph and Suggested Segments report are illustrated in Figures 31 and 32, and the finalized logical design is illustrated in Figure 33. The process is not finished until adequate support of the original data requirements is verified. In this case, they are supported, and the logical design process is concluded.

When doing the physical design, the designer may receive additional clues from the Association Weights report. This report shows the requirement to go in both directions or only in one

Figure 25 Parent/child graph

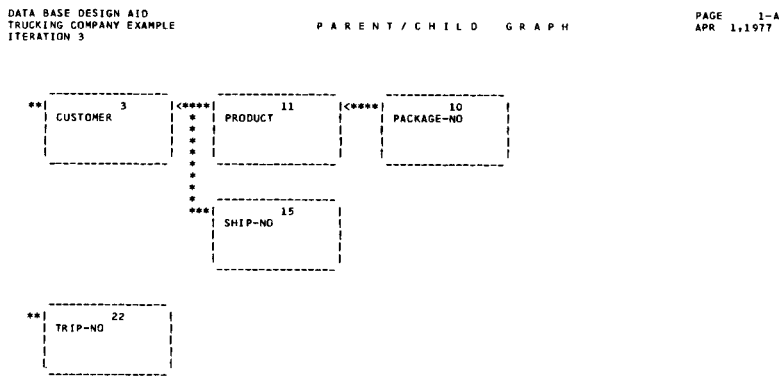


Figure 26 Suggested segments

DATA BASE DESIGN AID
TRUCKING COMPANY EXAMPLE
ITERATION 3

S U G G E S T E D S E G M E N T S

PAGE 1
APR 1,1977

KEY OF SEGMENT	F/V	LENGTH	DATA FIELDS (C=CONDITIONAL, I=INTERSECTION)
3 CUSTOMER	F	0+	LOCATION
10 PACKAGE-NO	F	0+	MFG, PACK-VOL, PACK-WT
11 PRODUCT	F	0+	DESCR
15 SHIP-NO	F	0+	
22 TRIP-NO	F	0+	SHIP-DATE, VEH-TYPE, VEH-VOL, VEH-WT

direction between parents of a logical relationship, suggesting the need for a bidirectional or unidirectional relationship. In addition, the association weights values, by indicating the relative traffic in both directions between the parents, may be helpful in determining virtual or physical pairing. In this case study, a bidirectional logical relationship is suggested between CUSTOMER and TRIP-NO.

iterative use

The iterative use of DBDA has been discussed. DBDA generates a logical design that functionally supports the data requirements but which may need refinement for performance or for other reasons. The user should also consider the iterative use of DBDA in another context.

Figure 27 Structural model

```
DATA BASE DESIGN AID                    S T R U C T U R A L    M O D E L                    PAGE     1
TRUCKING COMPANY EXAMPLE                                                                    APR  1,1977
ITERATION 3

                                                                FREQUENCY          WEIGHT
        KEY OF SEGMENT                  ADDITIONAL PARENTS      OF OCCURRENCE      C/P          P/C

3 CUSTOMER                                                      00001

    11 PRODUCT                                                  00020          0.000E+00    1.376E+04

        10 PACKAGE-NO                                           00003          1.590E+02    4.128E+04
                                        NE CUSTOMER                            0.000E+00    1.290E+02
                                            SHIP-NO                            4.128E+04    3.000E+01

    15 SHIP-NO                                                  00000          1.000E+00    0.000E+00
                                            TRIP-NO                            1.000E+00    0.000E+00

        10 PACKAGE-NO                                           00030          4.128E+04    3.000E+01
                                        NE CUSTOMER                            0.000E+00    1.290E+02
                                            PRODUCT                            1.590E+02    4.128E+04


DATA BASE DESIGN AID                    S T R U C T U R A L    M O D E L                    PAGE     2
TRUCKING COMPANY EXAMPLE                                                                    APR  1,1977
ITERATION 3

                                                                FREQUENCY          WEIGHT
        KEY OF SEGMENT                  ADDITIONAL PARENTS      OF OCCURRENCE      C/P          P/C

22 TRIP-NO                                                      00000

    15 SHIP-NO                                                  00000          1.000E+00    0.000E+00
                                            CUSTOMER                           1.000E+00    0.000E+00

        10 PACKAGE-NO                                           00030          4.128E+04    3.000E+01
                                        NE CUSTOMER                            0.000E+00    1.290E+02
                                            PRODUCT                            1.590E+02    4.128E+04
```

Figure 28 Candidates for secondary indexes

```
DATA BASE DESIGN AID                CANDIDATES FOR SECONDARY INDEXES                    PAGE     1
TRUCKING COMPANY EXAMPLE                                                                APR  1,1977
ITERATION 3

    *NOTE- THESE ASSOCIATIONS WERE SPECIFIED IN THE REQUIREMENTS BUT NEED SPECIAL ATTENTION TO IMPLEMENT.
          (FROM)                                (TO)
      INDEXING FIELD                        INDEXED FIELD

SHIP-DATE                           TRIP-NO
SHIP-NO                             CUSTOMER
SHIP-NO                             PACKAGE-NO
SHIP-NO                             SHIP-DATE
SHIP-NO                             TRIP-NO
SHIP-NO                             VEH-VOL
SHIP-NO                             VEH-WT
```

Rather than collecting all the data requirements and supplying the entire collection to DBDA initially, the user may find it advantageous to select two or three of the more important application functions, and process their requirements iteratively until a clean design is obtained. He may then add the requirements for some additional (but less important) functions, reprocess, and note where the resulting design differs from the one originally

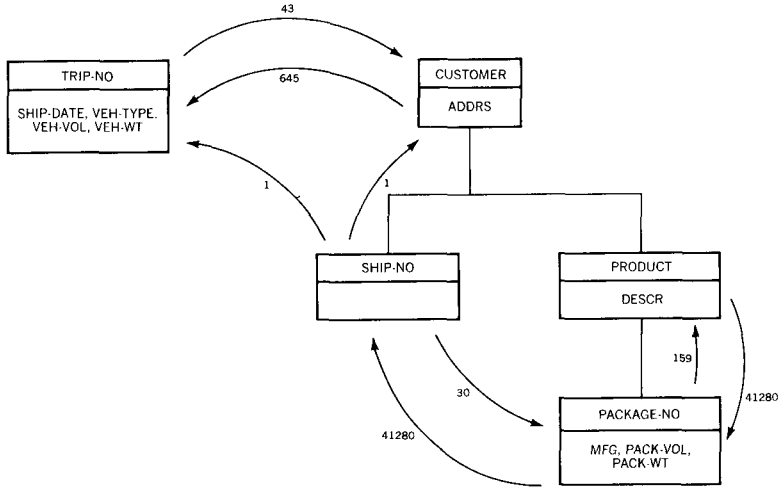
Figure 29 Association weights summary

DATA BASE DESIGN AID — ASSOCIATION WEIGHTS SUMMARY — PAGE 1
TRUCKING COMPANY EXAMPLE — (NORMALIZING PERIOD: MONTH) — APR 1,1977
ITERATION 3

ASSOCIATION (FROM)	(TO)	NORMALIZED BATCH WEIGHT	NORMALIZED ONLINE WEIGHT	TOTAL
CUSTOMER	PACKAGE-NO	1.290E+02	0.000E+00	1.290E+02
CUSTOMER	PRODUCT	0.000E+00	1.376E+04	1.376E+04
CUSTOMER	TRIP-NO	0.000E+00	6.450E+02	6.450E+02
PACKAGE-NO	MFG	3.000E+01	0.000E+00	3.000E+01
PACKAGE-NO	PACK-VOL	3.000E+01	0.000E+00	3.000E+01
PACKAGE-NO	PACK-WT	3.000E+01	0.000E+00	3.000E+01
PACKAGE-NO	PRODUCT	1.590E+02	0.000E+00	1.590E+02
PACKAGE-NO	SHIP-DATE	0.000E+00	4.128E+04	4.128E+04
PACKAGE-NO	SHIP-NO	0.000E+00	4.128E+04	4.128E+04
PRODUCT	PACKAGE-NO	0.000E+00	4.128E+04	4.128E+04
SHIP-DATE	TRIP-NO	1.075E+02	0.000E+00	1.075E+02
SHIP-NO	CUSTOMER	1.000E+00	0.000E+00	1.000E+00
SHIP-NO	PACKAGE-NO	3.000E+01	0.000E+00	3.000E+01
SHIP-NO	SHIP-DATE	1.000E+00	0.000E+00	1.000E+00
SHIP-NO	TRIP-NO	1.000E+00	0.000E+00	1.000E+00
SHIP-NO	VEH-VOL	1.000E+00	0.000E+00	1.000E+00
SHIP-NO	VEH-WT	1.000E+00	0.000E+00	1.000E+00
TRIP-NO	CUSTOMER	4.300E+01	0.000E+00	4.300E+01
TRIP-NO	SHIP-DATE	0.000E+00	6.450E+02	6.450E+02
TRIP-NO	VEH-TYPE	1.075E+02	0.000E+00	1.075E+02
TRIP-NO	VEH-VOL	1.075E+02	0.000E+00	1.075E+02
TRIP-NO	VEH-WT	1.075E+02	0.000E+00	1.075E+02

END OF REPORT

Figure 30 Two sets of candidates for logical relations



obtained. This enables the user to focus attention on those areas needing reconsideration or more detailed analysis, and if changes are required, to intelligently judge whether to change the specifications of the "more important" functions or of the functions just added to the study.

Summary

A technique has been described that can be embodied in a programming tool for reducing the labor of logical data base design.

Figure 31 New parent/child graph

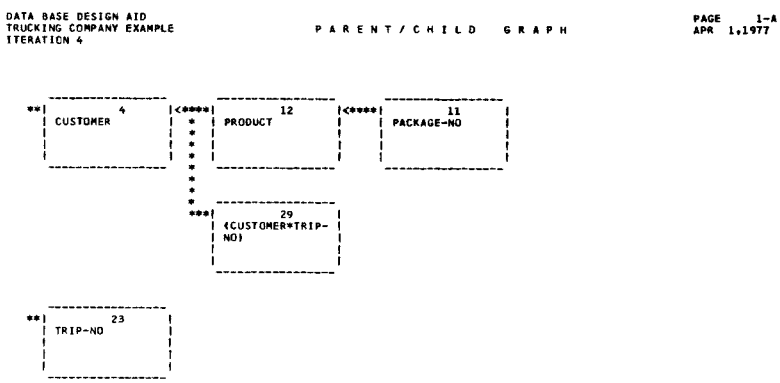
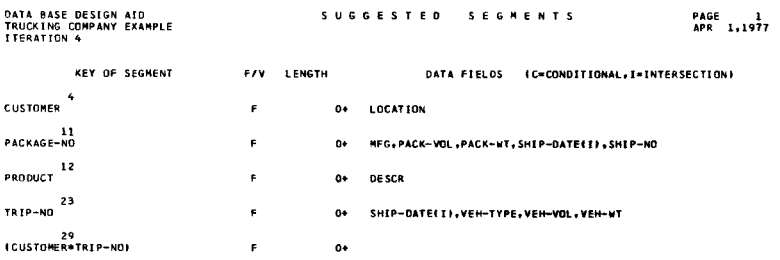


Figure 32 New suggested segments

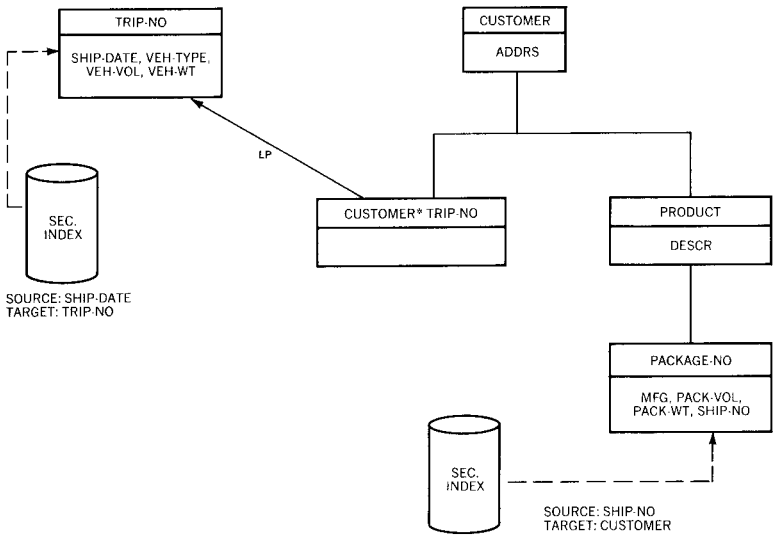


The techniques must be supplemented with the network restraints of the data base handler being used. For DL/I structures, the technique has been implemented in a program product called Data Base Design Aid (DBDA).

DBDA is a design aid in two senses. In one sense, it is a tool for analyzing the data requirements and suggesting a logical design of the data base. In another sense, it is a quality control tool for the experienced designer.

In analyzing the data requirements and generating the logical design, DBDA performs computations that generally are beyond the capabilities of manual analysis. Regardless of his level of experience, the designer is assisted by having this laborious work done for him. The designer still controls the design pro-

Figure 33 Finalized logical design



cess, but with DBDA, he is guided toward the design to be implemented. For the inexperienced designer, this guidance is a special advantage.

For the experienced designer who feels he can do an adequate design manually, DBDA serves as a quality control function to ensure that the design is consistent with the algorithms that should be followed in producing a design, and to ensure that none of the alternatives and design information that DBDA can report are overlooked.

Initial experiences with automated logical data base design as embodied in DBDA indicate that it can, indeed, be used as an aid for improving the quality of the design and for shortening the design and implementation time of a data base and its applications.

CITED REFERENCES

1. *IMS/VS Version 1, General Information Manual*, GH20-1260, IBM Corporation, Data Processing Division, White Plains, NY.
2. *DL/I DOS/VS, General Information Manual*, GH20-1246, IBM Corporation, Data Processing Division, White Plains, NY.
3. *Data Base Design Aid, General Information Manual*, GH20-1626; *Data Base Design Aid Program*, 5784-XX4; IBM Corporation, Data Processing Division, White Plains, NY.
4. C. T. Baker, *Inherent Structures in Data*, Technical Report TR 21.545, IBM Corporation, Kingston, NY (April 1974).
5. P. Suppes, *Introduction to Logic*, 229–240, D. Van Nostrand Co., Inc., Princeton, NJ (1957).

6. *Data Base Design Aid, Program Reference/Operations Guide*, SH20-1651, IBM Corporation, Data Processing Division, White Plains, NY.
7. *Data Base Design Aid, Designer's Guide*, GH20-1627, IBM Corporation, Data Processing Division, White Plains, NY.

GENERAL REFERENCES

1. C. J. Date, *An Introduction to Data Base Systems*, Addison-Wesley Publishing Co., Inc., Reading, MA (1975).

2. *Information Management System, System/Application Design Guide for IMS*, SH20-9025, IBM Corporation, Data Processing Division, White Plains, NY.

3. W. C. McGee, "The information management system IMS/VS," *IBM Systems Journal* **16,** No. 2, 84–168 (1977).
4. N. Raver and G. U. Hubbard, "Automated logical file design," *ACM Conference on Very Large Data Bases*, Framingham, MA (September 1975).

Reprint Order No. G321-5055.