AN INTRODUCTION TO DATA BASE NORMALIZATION

Richard L. Seltzer
Prudential Reinsurance Company

PREFACE

The terminology of relational theory is not strictly adhered to in this paper. Wherever possible, IMAGE terminology is used in order to facilitate understanding by the reader. Consequently, this paper cannot be considered a formal exposition of normalization.

I would like to acknowledge the cooperation and support of Prudential Reinsurance during the preparation of this paper. Thanks also go to Dean Livingston, Linda Mitchell, and Zoe Putnam for invaluable critiques on various facets of this paper, and especially to Lisa Masarek for presenting the initial opportunity.

INTRODUCTION

Relational data bases have been getting much attention over the past several years. A major feature of relational data base theory is normalization of data. Hierarchical data bases and network data bases (of which IMAGE is a subset) can benefit from the techniques of data normalization in the conceptual design of the data structure, as one facet of entity relationship analysis. This paper will describe and illustrate the several forms of normalization, discuss their implementation in IMAGE data bases, and examine the tradeoffs of such an implementation.

The examples used in this paper are very simple and the illustrated result of normalization may not differ much from the way that most people would design a data base. The purpose of this paper is, however, to illustrate the rules of normalization so that in more complex cases, the approach to normalization would be understood and a methodology applied to data base design.

I. NORMALIZATION DEFINED

Normalization is a method of organizing the data in a data base and decomposing the data into small, simple, non-redundant related units which contain functionally related data. Six forms of normalization are discussed here1. They are first (1NF), second (2NF), third (3NF), Boyce/Codd (BCNF), fourth (4NF), and fifth (5NF) normal forms.2 The higher normal forms are more strongly decomposed than the lower normal forms and are inclusive of them.

There are several advantages that result from normalization of data that may be missed in unnormalized data structures. Many affect the data and the database itself:

* Data base storage requirements are kept to a minimum. The elimination of redundant data also eliminates the need for space that the redundant data would have required.
* Maintenance of data is both eased and speeded. When fields are updated, they need be updated in only one place, not in many, due to the lack of redundancy.
* Inconsistency of data is avoided. Because, in most cases, there is only one copy of data, multiple copies of data with differing values does not occur.

* Currency of data is assured. Since there are not multiple copies of inconsistent data, there is no question as to which copy is the most current one.

* Data base manipulation is facilitated. The functional grouping of like data in one place, rather than in several places, eliminates major restructuring of the data base when fields or data sets are added or deleted.

Another advantage is that the data become more useful. Ad hoc data structures may be good for the requirement at hand, but should new functions be required, these data structures can become quite cumbersome. Since normalized data is kept in small, functional blocks, there exists a flexibilty allowing for use in new and different functions.

The key to normalization's advantages, decomposition of data into functional groups, also results in its disadvantage: an application may have to perform more retrieval operations because the data required resides in several data sets.3

These advantages and disadvantages will be illustrated throughout this paper.

II. THE NORMALIZATION PROCESS

Before detailing the normalization process, two concepts need to be explained: keys and functional dependence.

A key is a value used to identify a record. (Do not confuse this key with the IMAGE search item, which is used for path implementation purposes.) A key may be one field, or it may consist of several fields, in which case it is known as a composite key. There are several types of keys.

A primary key is a value that uniquely identifies a record within a data set. Every record in a normalized data base must have a primary key. An example of a single field primary key is Part-no as the key in data set PARTS (which describes a particular part). A composite primary key would be Part-no and Warehouse-no as the key in data set PART-WAREHOUSE (which gives the quantity of a particular part at a particular location). A primary key value, or any part of it (as in a composite primary key), may not be null. If it were, it could not uniquely identify the record. Primary keys in the examples in this paper are denoted by underlining.

Candidate keys are multiple keys in a record, each of which uniquely identifies that record. For example, if each supplier has a unique name as well as a unique number, both Supplier-no and Supplier-name are candidate keys. Candidate keys in the examples in this paper are denoted by a pound sign (#).

A foreign key is a non-primary key value in a data set that is also a primary key in another data set. The individual values within a composite primary key are also considered foreign keys. A foreign key can be used to represent a relationship between data sets. For example, in a case where a particular shipper will be shipping a part on order from a certain supplier, the PARTS- SUPPLIER data set would contain a field for Shipper-no to indicate who will be shipping that part. All the information about that shipper would be found in the SHIPPERS data set under the primary key Shipper-no. Because they are needed to maintain relationships, foreign keys are the only redundant data found in a fully normalized data base. Foreign keys in the examples in this paper are denoted by an asterisk (*).

Functional dependence deals with the identification of the value of one field by another field or key. Field B is functionally dependent on field A if and only if for any A value there exists only one B value. For example, given Supplier-no (field A), you can determine the Supplier-address (field B); Supplier-address is functionally dependent on Supplier-no. Field A is also known as the determinant. Since the primary key uniquely identifies a record, the primary key is the determinant of all of the fields in that record.

The process of normalizing data will now be illustrated in a stepwise fashion.4 As an example, let's use a parts inventory system. The information necessary for this system, along with an explanation of the fields, is represented in Figure 1.

```
Part-no              -- identifies a part
Part-cost            -- cost of the part
Parts-on-hand        -- quantity available of the part
Parts-on-hand-value  -- total inventory value of the part
```

```
Supplier-no-1           -- identifies a supplier of the part
Supplier-name-1         -- supplier's name
Supplier-address-1      -- supplier's address
Supplier-bal-due-1      -- total balance due the supplier for all orders
Parts-on-order-1        -- quantity of the part on order from the supplier
Shipper-no-1            -- identifies the shipper of the part on order
Shipper-name-1          -- shipper's name
Shipper-address-1       -- shipper's address
Supplier-no-2           -- identifies a second supplier of the part
Supplier-name-2         -- second supplier's name
Supplier-address-2      -- second supplier's address
Supplier-bal-due-2      -- total balance due second supplier for all orders
Parts-on-order-2        -- quantity of the part ordered from this supplier
Shipper-no-2            -- shipper of part on order from second supplier
Shipper-name-2          -- shipper's name
Shipper-address-2       -- shipper's address
Warehouse-no-1          -- identifies warehouse where part is stored
Warehouse-loc-1         -- location of the warehouse
Warehouse-capacity-1    -- capacity of the warehouse
Wh-parts-quantity-1     -- quantity of the part stored in the warehouse
Wh-manager-no-1         -- identifies the manager of the warehouse
Wh-manager-name-1       -- manager's name
Wh-manager-salary-1     -- manager's salary
Warehouse-no-2          -- another warehouse where the part is stored
Warehouse-loc-2         -- location of second warehouse
Warehouse-capacity-2    -- capacity of second warehouse
Wh-parts-quantity-2     -- quantity of the part stored in second warehouse
Wh-manager-no-2         -- identifies manager of the second warehouse
Wh-manager-name-2       -- manager's name
Wh-manager-salary-2     -- manager's salary
```

FIGURE 1
Unnormalized Data

This data organization is prone to wasted space. For example, a part can be supplied by many suppliers. If a particular part is supplied by only one supplier, the information relevant to that supplier will be put in Supplier-no-1, Supplier-name-1, Supplier-address-1, and Supplier-bal-due-1. The fields for Supplier-2 (and however many more suppliers there are) will be empty.5

First normal form resolves this problem by removing the repetitious fields. A data set is in 1NF if and only if it has no repeating fields. The use of IMAGE sub-items would violate 1NF.

In order to place our data into 1NF, the Supplier, Shipper, Warehouse and other repititious fields must be replaced by single sets of fields. The data after this transformation, now in 1NF, appear in Figure 2.

Each record in this data set, which has been labeled GENERAL-PART-INFO, can be uniquely identified (i.e., determined) by a composite key made up of Part-no, Supplier-no, and Warehouse-no. Every field is filled6, and there is no wasted space.

```
GENERAL-PART-INFO

Part-no
Part-cost
Parts-on-hand
Parts-on-hand-value
Supplier-no
Supplier-name
```

```
Supplier-address
Supplier-bal-due
Parts-on-order
Shipper-no
Shipper-name
Shipper-address
Warehouse-no
Warehouse-loc
Warehouse-capacity
Wh-parts-quantity
Wh-manager-no
Wh-manager-name
Wh-manager-salary
```

FIGURE 2
Data in 1NF

There are still problems with this form, though. For example, if a supplier is not currently supplying a part, or a warehouse is not currently being used, information about that supplier or warehouse will not be in the data base until a part is supplied by that supplier or a part is stored in that warehouse. Similarly, if only one part is supplied by a particular supplier, and it is no longer purchased from him, the information about that supplier is lost. Keeping the supplier or warehouse information in a record with a null Part-no would not be a solution; that would violate the rule that no value in a primary key may be null.

Another problem arises with the information that is in this data set. It is redundant. If a supplier supplies three different parts, information about that supplier, such as the address, appears three times. This creates extra work for updating. Whenever the address changes, every record in which it appears must be updated. If not every record is changed, the data base becomes inconsistent, leading to other troubles, such as determining which datum is the most current.

The logical solution to these problems, and the idea behind second normal form, is to somehow extract this redundant information about a specific entity and deposit a single copy of it somewhere else. This is accomplished by creating data sets containing information which is dependent on the entire primary key and not on only part of the (composite) primary key. (This type of functional dependence is called full functional dependence.) For example, the composite key Part-no Warehouse-no determines the quantity of a particular part in a particular warehouse. A field such as part-cost is functionally dependent only on Part-no and not on Warehouse-no and is thereby not fully functionally dependent on the entire key Part-no Warehouse-no.

A data set is in 2NF if and only if it is in 1NF and every nonkey field is fully dependent on the primary key.

To decompose GENERAL-PART-INFO into 2NF, the fields that are only partially dependent on the primary key will be removed to separate data sets with a primary key on which they are fully dependent. The original primary key does remain in the original data set, and its components function as foreign keys. In our example, six new data sets will be created, as appear in Figure 3, and all of the fields in them are fully functionally dependent on their respective primary keys. The data set GENERAL-PART-INFO, renamed PART-SUPPLIER-WAREHOUSE after the decomposition, denotes the relationship among parts, suppliers of those parts, and warehouses in which those parts are supplied.

```
PARTS                SUPPLIERS            WAREHOUSES

Part-no              Supplier-no          Warehouse-no
Part-cost            Supplier-name#       Warehouse-loc
Parts-on-hand        Supplier-address     Warehouse-capacity
Parts-on-hand-value  Supplier-bal-due     Wh-manager-no
                                          Wh-manager-name
                                          Wh-manager-salary
```

PART-WAREHOUSE	PART-SUPPLIER	PART-SUPPLIER-WAREHOUSE
Part-no*	Part-no*	Part-no*
Warehouse-no*	Supplier-no*	Supplier-no*
Wh-parts-quantity	Parts-on-order	Warehouse-no*
	Shipper-no	
	Shipper-name	
	Shipper-address	

FIGURE 3
Data in 2NF

Even though there are now six data sets instead of one, the total information in them is preserved through relationships expressed by the foreign keys. For example, from the data set PART-WAREHOUSE, which contains the quantity of a particular part stored in a particular warehouse, information pertaining uniquely to the part can be obtained by taking Part-no and finding the record in the PARTS data set containing the same Part-no. The same thing can be done with Warehouse-no and the WAREHOUSE data set.7

Despite the fact that the data sets are in 2NF, there are still some redundancy problems. For example, a shipper may have information about him recorded in several places if he is shipping several parts, leading to data consistency and currency questions; information about the shipper will be lost if he is not shipping any parts at this time.

These problems arise because the data in question is information not about the primary key, but about another non-key field. The data does not directly describe (or depend on) the primary key, but does so transitively through the non-key field. Take as an example the data set WAREHOUSES. Wh-manager-salary is an attribute of Wh-manager-no (the manager earns a specific salary) and is directly dependent on it. Wh-manager-no is an attribute of Warehouse-no (the warehouse has one manager) and is directly dependent on it. Consequently, for a warehouse, there is only one manager-salary paid, and Wh-manager-salary is therefore transitively dependent on Warehouse-no by way of Wh-manager-no.

The solution to the redundancy problems is to extract this transitively dependent information and place it in its own data set. This is accomplished by third normal form.

A data set is in 3NF if and only if it is in 2NF and every nonkey field is directly dependent on the primary key.

Decomposition of 2NF data sets into 3NF data sets is done by removing to their own data set the non-key data that directly describe other non-key data. The primary key of the new data set is the "directly-described" non-key data. A copy of this key remains in the original data set to serve as a foreign key.

To decompose the data in Figure 3 into 3NF, as represented in Figure 4, three new data sets, SHIPPERS, MANAGERS, and PART-VALUES8, are created. (Note that the fields Part-cost and Parts-on-hand form a composite primary key in the data set PART-VALUES. Therefore, these two fields together form a composite foreign key in the data set PARTS.)

(3NF does have a weakness in that it does not satisfactorily handle cases of two or more composite and overlapping keys.9 As a result of this shortcoming, 3NF has been replaced by Boyce/Codd normal form, which is stronger yet conceptually simpler. A data set is in BCNF if and only if every determinant is a candidate key. This means that every field in the data set must be directly functionally dependent on the entire primary key, and only candidates of the entire primary key are allowed. As you can see, Figure 4 is also in BCNF.10)

PARTS	SUPPLIERS	WAREHOUSES
Part-no	Supplier-no	Warehouse-no
Part-cost*	Supplier-name#	Warehouse-loc
Parts-on-hand*	Supplier-address	Warehouse-capacity
	Supplier-bal-due	Wh-manager-no*

PART-VALUES	PART-WAREHOUSE	PART-SUPPLIER
Parts-on-hand	Part-no*	Part-no*
Part-cost	Warehouse-no*	Supplier-no*
Parts-on-hand-value	Wh-parts-quantity	Parts-on-order
		Shipper-no*

SHIPPERS	MANAGERS	PART-SUPPLIER-WAREHOUSE
Shipper-no	Manager-no	Part-no*
Shipper-name#	Manager-name#	Supplier-no*
Shipper-address	Manager-salary	Warehouse-no*

center 2 FIGURE 4 Data in 3NF/BCNF

Fourth normal form deals with multivalued dependencies. A multivalued dependence (MVD) can be considered a special case of functional dependence, but instead of one specific field value associated with a given key, there is a specific set of values for that key.

Putting two or more MVD's that are independent of each other (i.e., they are dependent on the primary key, but have no effect on each other) in the same data set leads to redundancy problems. For example, a shipper can make deliveries in one or more time periods -- day, night, or weekend. The shipper can also handle, independent of delivery time, one or more kinds of freight -- regular, fragile, electronic, or perishable -- and have a rating for each type. One way of arranging this data is as in Figure 5. This data set is in BCNF -- the entire record is the primary key. Still, redundancy problems remain. Add another type of freight for a shipper, and many records, not just one, will have to be added to the data set. Updating of records will also suffer from the effects of data redundancy. Independent MVD's will have to be put in separate data sets to avoid these problems.

Shipper-no	Delivery-time	Freight-type
1	D	R
1	D	F
1	D	E
1	D	P
1	N	R
1	N	F
1	N	E
1	N	P
1	W	R

1	W	F
1	W	E
1	W	P
2	D	R
2	D	F
2	D	P
2	N	R
2	N	P
2	W	R
2	W	P
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
.	.	.
143	D	R
143	D	F
143	N	R
143	N	F
143	W	R

FIGURE 5
MVD's in BCNF

A data set is in 4NF if and only if it does not contain more than one independent MVD about an entity.

To decompose a data set into 4NF, each MVD is paired with the primary key it is dependent upon to form a new composite primary key for a new data set. Any information that is fully and directly dependent on this new primary key is removed from the original data set and placed in the new data set.

If, in our data base, we want to keep information about a shipper's delivery times, the types of freight he carries, and, additionally, his rating for that type of freight, the data will be decomposed into two small data sets11:

Shipper-no Delivery-time &
Shipper-no Freight-type Rating

The full data base in 4NF, with these new data sets, appears in Figure 6.

PARTS	SUPPLIERS	WAREHOUSES
Part-no	Supplier-no	Warehouse-no
Part-cost	Supplier-name#	Warehouse-loc

Parts-on-hand	Supplier-address Supplier-bal-due	Warehouse-capacity Wh-manager-no*
PART-WAREHOUSE	PART-SUPPLIER	PART-SUPPLIER-WAREHOUSE
Part-no* Warehouse-no* Wh-parts-quantity	Part-no* Supplier-no* Parts-on-order Shipper-no*	Part-no* Supplier-no* Warehouse-no*
SHIPPERS	MANAGERS	
Shipper-no Shipper-name# Shipper-address	Manager-no Manager-name# Manager-salary	
SHIPPER-TIMES	SHIPPER-FREIGHT	
Shipper-no* Delivery-time	Shipper-no* Freight-type Rating	

FIGURE 6
Data in 4NF

Fifth normal form provides a general purpose decomposition. It is conceptually based not only on the determination of fields in the data set, as are the other normal forms, but also on the values within the data set. Consequently, it can be difficult to explain 5NF and to identify 5NF data sets. Therefore, what follows is only a brief overview.

5NF, like 4NF, deals with multiple MVD's, but in cases where they are not independent of each other. Take the PART-SUPPLIER-WAREHOUSE data set created as a result of 2NF. It contains MVD's, since a part could come from many suppliers and could be stored in many warehouses. These MVD's are not, however, independent of each other. All of a supplier's parts are not necessarily in all of the warehouses.

A data set is in 5NF if and only if it cannot be further decomposed without causing the creation of spurious information upon reconstruction.

For a non-5NF data set to be decomposed into 5NF, it must meet a certain constraint. Using our example, it can be stated like this: if a part

is stored in certain warehouses, and that part is obtained from certain suppliers, then the part from each and every supplier will be in each and every warehouse.12

When this constraint holds, a data set with dependent MVD's can be decomposed into three or more (not two, as in the previously mentioned normal forms) smaller data sets, from which the original information can be accurately reconstructed by combining all of these data sets if the original data set meets the specific constraint. However, when the constraint does not hold, a recombination of the smaller data sets results in extra, spurious information.

Figure 7 illustrates a case where the data set cannot be decomposed and accurately recreated. PART-SUPPLIER-WAREHOUSE is decomposed into three smaller data sets PART-SUPPLIER, SUPPLIER-WAREHOUSE, and PART-WAREHOUSE (the PART-WAREHOUSE data set already existing in our data base serves this purpose). When PART-SUPPLIER and SUPPLIER-WAREHOUSE are joined by the common value of their common key, Part-no, and the result joined with PART-WAREHOUSE by the common values of their common keys Part-no and Warehouse-no, the reconstructed data set contains an extra, spurious record (P1, S1, W1). (Spurious records are indicated in the example by an exclamation point (!).) Therefore, the original PART-SUPPLIER-WAREHOUSE data set is in 5NF.

If the record (P1, S1, W1) were in the PART-SUPPLIER-WAREHOUSE data set, it could be accurately reconstructed from the decomposed data sets. In that case, PART-SUPPLIER-WAREHOUSE would not be in 5NF, but would have to be decomposed into the three smaller 5NF data sets.

ORIGINAL DATA SET

Part-no	Supplier-no	Warehouse-no
P1	S1	W2
P1	S2	W1
P2	S1	W1

DECOMPOSED DATA SETS

Part-no	Supplier-no
P1	S1
P1	S2
P2	S1

Supplier-no	Warehouse-no
S1	W1
S1	W2
S2	W1

Part-no	Warehouse-no
P1	W1
P1	W2
P2	W1

RECOMPOSITION (INTERMEDIATE STEP)

	Part-no	Supplier-no	Warehouse-no
!	P1	S1	W1
	P1	S1	W2
	P1	S2	W1
	P2	S1	W1
!	P2	S1	W2

Part-no	Warehouse-no
P1	W1
P1	W2
P2	W1

RECOMPOSED DATA SET WITH SPURIOUS RECORD

Part-no	Supplier-no	Warehouse-no

P1	S1	W1
P1	S1	W2
P1	S2	W1
P2	S1	W1

FIGURE 7
Dependent MVD Decomposition and Recomposition

III. IMAGE IMPLEMENTATION

We now have our data base decomposed into several small, functionally cohesive, nonredundant data sets. A logical view of these data sets and how they relate to each other is presented in Figure 8. The arrows represent whether a one-to-one, one-to-many, or many-to-one relationship exists between data sets.

As you can see, there are several levels of dependence. IMAGE, however, being only a two level network data base management system, can efficiently handle only one dependence level for one-to-many relationships, making hierarchy implementation awkward.[13] It also has, like any data base management system, implementation features to consider. How do we bring the logical data base into conformity with the real world of IMAGE? Here are some general rules (of course, general rules always have exceptions):

1. Data sets that have a single-field primary key, such as the data set PARTS, should be manual master data sets. This way, a search for a value, for example, that of a foreign key in another data set, can hash directly to the proper record.

2. Data sets that have a composite primary key, such as the data set PART- SUPPLIER, should be detail data sets. A composite key implies a one-to- many relationship between master data sets having single-field keys and the composite-key detail data set itself; consequently, paths between these data sets can be established. Automatic master data sets can be established to create paths for key fields without corresponding master data sets (for example, an automatic master for FREIGHT with a path to SHIPPER-FREIGHT) or for non-key fields for which quick access is a necessity.

3. Those manual master data sets requiring quick access of non-key fields can be dropped a level to detail data sets, and automatic master data sets can be created to establish the search paths. (I must state here the standard warnings about many paths in a detail data set: while retrieval is fast, updating is slow, and maintenance of the data base can be difficult.)

The IMAGE-implemented normalized data base is shown in Figure 9. Standard HP graphic representation is used.

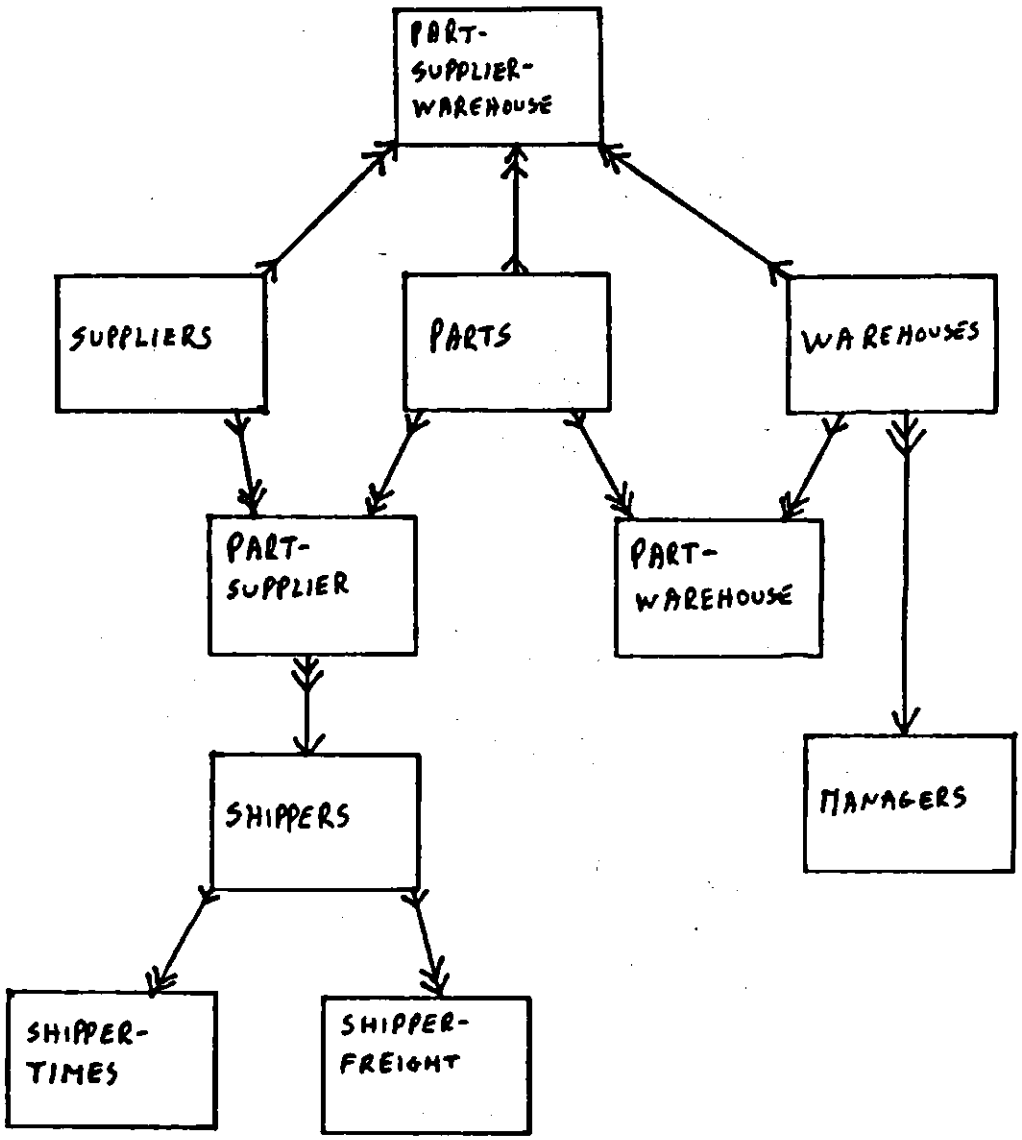

Figure 8
Logical Relationships Among the Data Sets

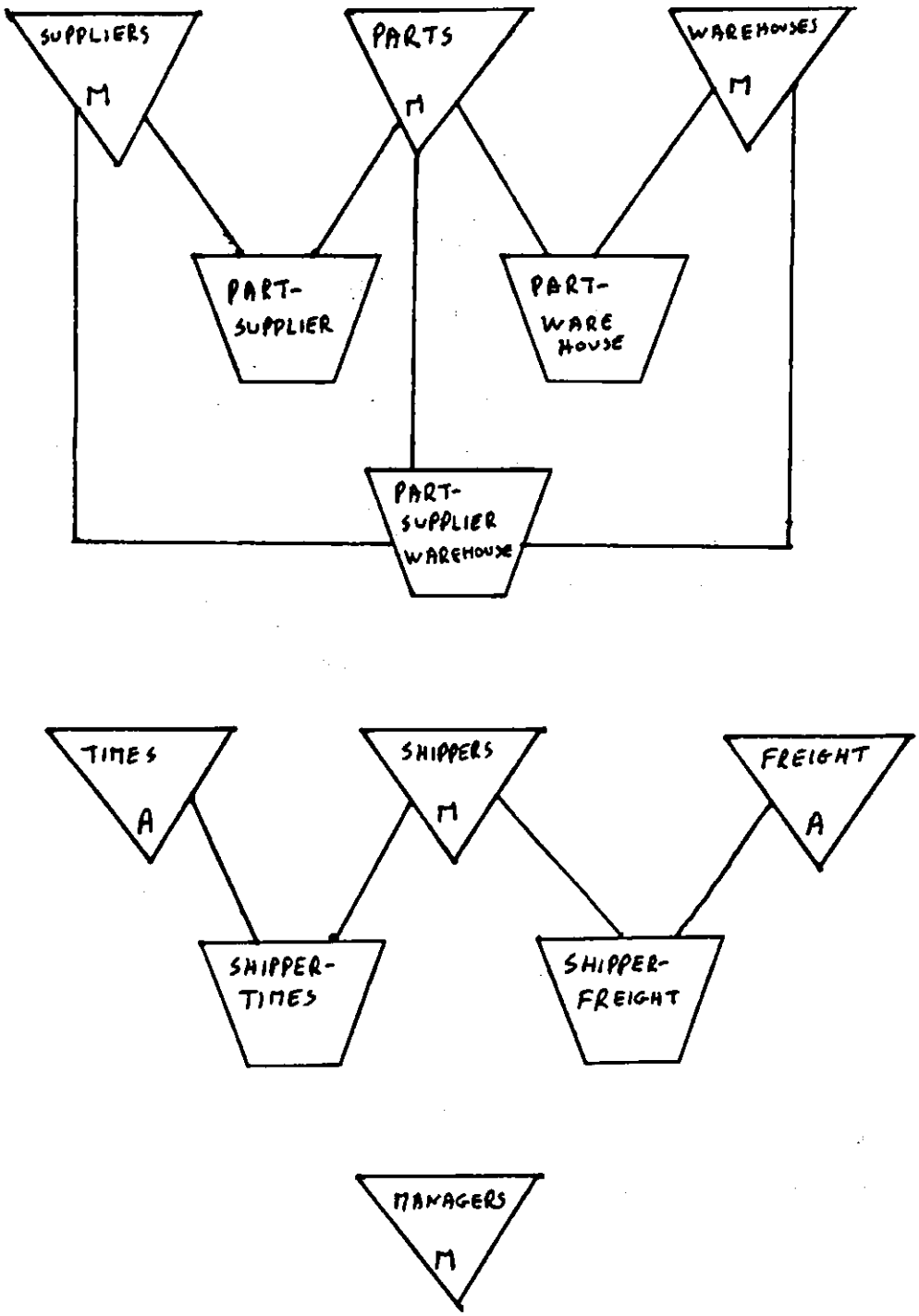

Figure 9
IMAGE Implementation of Data Base

Figure 9
IMAGE Implementation of Data Base

CONCLUSION

Normalization of data provides many advantages. The data base requires much less space than an unnormalized data base would. Because of the nonredundancy of data, the data base becomes more useful and can be maintained much more easily, both physically (except for IMAGE search items, which are often the redundant keys) and logically (as shown in the examples).

On the other hand, application programs must retrieve data from many data sets instead of from only one, thereby increasing retrieval time. The question then arises, how many levels of normalization to apply to the data before the cost of extended retrieval time overshadows the benefits of normalization. The answer is application dependent, but it would be difficult to justify not commonly normalizing to at least 3NF/BCNF.

NOTES

1. There are other normal forms, such as (3,3) normal form and domain-key normal form. These, however, are beyond the scope of this paper.

2. 1NF, 2NF, and 3NF were discovered by E. F. Codd (see Codd[1970] and Codd[1972]). BCNF was discovered by Codd and R. F. Boyce (see E. F. Codd, "Recent Investigations into Relational Database Systems." Proc. IFIP Congress 1974.). 4NF ad 5NF were discovered by R. Fagin (see R. Fagin, "Multivalued Dependencies and a New Normal Form for Relational Databases." ACM Transactions on Database Systems. 2, No. 3 (September 1977). and R. Fagin, "Normal Forms and Relational Database Operators." ACM SIGMOD International Conference on Management of Data.).

3. To make the data base structure transparent to the applications, shield the applications from data base maintenance, and prevent excessive program maintenance, an access module might be used. When the structure of the data changes, only the access module needs to be changed, not the application programs. The access module itself would add slightly to retrieval time.

4. It is interesting to compare stepwise normalization of data into nonredundant functional groups with structured programming's idea of stepwise refinement of programs, as presented by Wirth and Dijkstra, and nonredundant functional procedures/modules. One could be called "structured data" and the other "'normalized' programs" (although there is no formal method of which I know for "normalizing" programs). (See Wirth, Systematic Programming (1973), Algorithms + Data Structures = Programs (1975) and Dijkstra Structured Programming (1972).)

5. This type of data organization resulted in a real-life problem on an IMAGE data base. A detail data set had a series of names, Name-1, Name-2, Name- 3, and Name-4, which were dependent upon the record key. All four names were not always entered. Sometimes only one name was entered, in Name-1; sometimes only two names were entered, in Name-1 and Name-2; and so on. Most of the time, no Name-4 was entered. Those name fields not entered were filled, by default, with blanks.

   These name fields were often used for searching this data set, so they were made search items and linked to an automatic master. Unfortunately, because of all the blank entries in Name-4, the maximum chain length was quickly reached for the blank values and no more records could be added to that data set unless all the name fields were filled.

6. That is, there is no systematic occurrence of sets of empty fields. Only fields that are part of a key are forbidden to be null. Non-key fields may contain null values, but these should be rarely occur.

7. When physically or logically reconstructing the data through the foreign keys, multiple combinations of the reconstructed records become possible. For example, given the data sets:

PARTS WAREHOUSES PART-WAREHOUSE

Part-no	Part-info
PN1	PI1...
PN2	PI2...

Warehouse-no	Warehouse-info
WN1	WI1...
WN2	WI2...
WN3	WI3...

Part-no	Warehouse-no
PN1	WN1
PN1	WN2
PN1	WN3
PN2	WN1
PN2	WN2
PN2	WN3

the following redundant data set would result from recombination:

PART-WAREHOUSE-COMBINED

Part-no	Warehouse-no	Part-info	Warehouse-info
PN1	WN1	PI1...	WI1...
PN1	WN2	PI1...	WI2...
PN1	WN3	PI1...	WI3...
PN2	WN1	PI2...	WI1...
PN2	WN2	PI2...	WI2...
PN2	WN3	PI2...	WI3...

Consequently, we see that the amount of space saved by decomposition is a multiplicative function. This concept is highlighted by the fact that the relational operator used for reconstructing data sets through common keys, join, is based on the traditional set operator Cartesian product.

8. PART-VALUES appears here for illustrative purposes. While it is in 3NF, it is really nothing more than a multiplication table. Since the Parts- on-hand-value field can be calculated, this data set is actually unnecessary and its presence contradicts the philosophy behind normalization. For this reason, the PART-VALUES data set will not appear subsequently in examples. Pragmatically, the relative importance of processing time to storage space is the deciding factor in whether data that can be calculated should be included in a data base.

9. As an example, take the following case:

Part-no Supplier-no Supplier-name Parts-on-order

Because Supplier-no and Supplier-name are candidate keys, this data set has two overlapping candidate primary keys: Part-no Supplier-no, and Part- no Supplier-name. This data set is in 3NF, since every non-key field is directly dependent on the primary key. However, there is still a possibility of redundancy in Supplier-name. This is because 3NF does not require a field to be fully dependent on the primary key if it is itself a component of a candidate key.

10. BCNF can sometimes be too strong, destroying the dependence relationships of the decomposed data sets. As an example, take the data set

Project Leader Salary

This data set is in 2NF but not 3NF nor BCNF. It is possible to decompose this data as such:

Project Leader & Project Salary

These two data sets are in BCNF, but the dependence relationship and the independence of the data have been lost, resulting in insertion, deletion, and update irregularities. For example, it is not possible to know a given project leader's salary unless that project leader is manging a project, and if he is managing two projects, his salary is recorded twice.

It is also possible to decompose the data this way:

Project Salary & Leader Salary

These, too, are in BCNF, but if we combine these two data sets (logically or physically) based on the common field Salary, extra, spurious records will be created. This results from the fact that many project leaders may earn the same salary, and the combined data set is a product of all possible combinations of the smaller data sets linked by common salaries.

The optimal decomposition, reflecting the hierarchy of the relationships, is this:

Project Leader* & Leader Salary

This structure shows which project leader manages a given project, and we know a project leader's salary even when he is not leading any project.

11. There are perfectly valid reasons for having one data set with all of the information, such as when you need information that is fully dependent on several multivalued dependencies (this is really a result of 2NF). For example, to record the on-time percentage of a given shipper at a given time for a specific type of freight, the following data set is needed:

Shipper Delivery-time Freight-type On-time-percentage

From the information in this data set, you can calculate the on-time- percentage for a single delivery-time or a single freight-type, or for any combination. If you don't want to calculate this information, you can build new data sets (e.g., Delivery-time Freight-type On-time- percentage) to house that information, but that saves calculation time at the expense of extra space consumption and data redundancy.

Note that even in this case, the data set

Shipper-no Freight-type Rating

will still exist to prevent redundancy within rating type. Of course, On-time-percentage could be a part of this data set, too, under the "save time, not space" option.

12. More formally, the constraint, called a join dependence, can be described this way:

If the records

(P1, S1, W2)
(P1, S2, W1)
(P2, S1, W1)

appear in the data set PART-SUPPLIER-WAREHOUSE, then the record

(P1, S1, W1)

appears in the data set PART-SUPPLIER-WAREHOUSE.

13. For example, if data set A has a one-to-many relationship to data set B, and data set B has a one-to-many relationship to data set C (not illus- trated in this paper), in the best case, retrieval of a record from data set C requires a chain traversal from master data set A to detail data set B, then a hash to master data set C', from which another chain traversal to detail data set C is done. One-to-one and many-to-one relationships are easily handled by hashing directly to the entry in a master data set.

BIBLIOGRAPHY

Codd, E.F. "A Relational Model of Data for Large Shared Data Banks." Communications of the ACM. 13, No. 6 (June 1970) 377-387.

----------. "Further Normalization of the Data Base Relational Model." Data Base Systems (Courant Computer Science Symposium 6). Ed. Randall Rustin. Englewood Cliffs, N.J.: Prentice-Hall, 1972. 33-64.

Date, C.J. An Introduction to Data Base Systems. Third Edition. Reading, Mass.: Addison-Wesley, 1981.

Flores, Ivan. Data Base Architecture. New York, N.Y.: Van Nostrand Reinhold, 1981.

IMAGE Data Base Management System Reference Manual. Second Edition. Cupertino, Ca.: Hewlett-Packard, 1979.

Kent, William. "A Simple Guide to Five Normal Forms in Relational Data Base Theory." Communications of the ACM. 26, No.2 (February 1983) 120-125.

Ullman, J.D. Principles of Database Systems. Second Edition. Rockville, Maryland: Computer Science Press, 1982.

Further references may be obtained from the excellent bibliographic notes in Date.

Richard Seltzer is familiar with a variety of hardware, having his most extensive experience on HP 3000's at the Prudential Reinsurance Co. in Newark, N.J. Among

his areas of involvement on the HP 3000 are data base design, the design and programming of both business and software systems (such as security and chargeback systems), and systems management. He is currently pursuing an M.S. in Computer Methodology from Baruch College of the City University of New York. For his thesis, he is writing a structured English query language based on relational algebra.
