# MANAGEMENT INFORMATION SYSTEMS - A COMPARISON OF THE NETWORK AND RELATIONAL MODELS OF DATA

by

Charles Arthur Ziering, Jr.

B.S., Massachusetts Institute of Technology

(1973)

Submitted in Partial Fulfillment of the

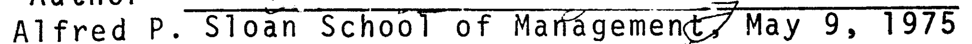
Requirements for the Degree of

Master of Science

at the

Massachusetts Institute of Technology

June, 1975



Signature of Author ______________________________
Alfred P. Sloan School of Management, May 9, 1975

Certified by ______________________________
Professor Stuart E. Madnick, Thesis Supervisor

Accepted by ______________________________
Chairman, Departmental Committee on Graduate Students

# MANAGEMENT INFORMATION SYSTEMS - A COMPARISON OF THE NETWORK AND RELATIONAL MODELS OF DATA

by

Charles Arthur Ziering, Jr.

Submitted to the Sloan School of Management May 9, 1975 in partial fulfillment of the requirements for the Degree of Master of Science.

## ABSTRACT

This paper compares the two predominant models of data underlying database management systems. After explaining the problems each is hoping to solve, each model is discussed as background. Five points of view with respect to a database management system are given, and the two models are compared in terms of the needs of each. A network algebra and network calculus are introduced to allow for comparison of the two models on the same level. Finally, a hybrid view is presented to handle distributed databases.

THESIS SUPERVISOR: Stuart E. Madnick

TITLE: Assistant Professor of Management

## ACKNOWLEDGEMENTS

Many of the ideas presented here grew out of "discussions" of the relative merits of the two models, primarily with Grant N. Smith. This forum was invaluable in clarifying the arguments.

As usual, the inciteful comments of and discussions with Professor Stuart E. Madnick were of immense help. I suspect that much of the credit for any success found in the presentation is due him.

My knowledge and understanding of the network model comes largely from my experience at MITROL, Inc., and discussions there with Jeffrey P. Stamen.

When the crunch came to get the manuscript prepared in time, the help of my typist, Norma Robinson, was greatly appreciated.

Table of Contents


Page

1. Introduction
   - The Use of Computers . . . . . . . . . . 7
   - Current Problem Areas . . . . . . . . . 10
   - The Reason for the Problems . . . . . . 13
   - A Logical View of Data . . . . . . . . . 15
   - History . . . . . . . . . . . . . . . . 17
   - The Current Conflict . . . . . . . . . . 20
   - Presentation Strategy . . . . . . . . . 23

2. The Network Model of Data
   - Introduction . . . . . . . . . . . . . . 25
   - Data versus Information . . . . . . . . 25
   - Entities, Attributes, and Keys . . . . . 26
   - Entity Classes . . . . . . . . . . . . . 27
   - Associations Between Entities . . . . . 28
   - Standard Terminology . . . . . . . . . . 36
   - Data Structure Diagrams . . . . . . . . 36
   - Trees and Networks . . . . . . . . . . . 37
   - Relation Naming . . . . . . . . . . . . 40

3. The Relational Model of Data
   - Introduction . . . . . . . . . . . . . . 43
   - Goals . . . . . . . . . . . . . . . . . 43
   - The Model . . . . . . . . . . . . . . . 46
   - First Normal Form . . . . . . . . . . . 47
   - Operations of Relations . . . . . . . . 50
   - Second and Third Normal Form . . . . . . 54
   - Algebra and Calculus . . . . . . . . . . 57

4. Viewpoints for Comparison
   - Introduction . . . . . . . . . . . . . . 59
   - Database Administrator/Designer . . . . 60
   - A Framework for Language Interface Comparison . . . . . . . . . . . . . 62
   - Experienced User . . . . . . . . . . . . 66
   - Casual User . . . . . . . . . . . . . . 68
   - The System . . . . . . . . . . . . . . . 69
   - Application Programs . . . . . . . . . . 72

Table of Contents (cont'd)


| | Page |
|---|---|
| 5. A Network Algebra | |
| Introduction . . . . . . . . . . . . . . . . | 73 |
| Access Path Navigation . . . . . . . . . | 73 |
| Extension to the Relational Algebra . | 74 |
| Examples . . . . . . . . . . . . . . . . . | 77 |
| 6. A Network Calculus | |
| Introduction . . . . . . . . . . . . . . . . | 80 |
| The Extension . . . . . . . . . . . . . . | 80 |
| The Network Calculus . . . . . . . . . . | 80 |
| Examples . . . . . . . . . . . . . . . . . | 84 |
| Reduction . . . . . . . . . . . . . . . . | 86 |
| Optimization and Efficiency . . . . . . | 87 |
| 7. Summary | |
| Conclusions . . . . . . . . . . . . . . . | 89 |
| A Hybrid View . . . . . . . . . . . . . . | 92 |
| Further Research . . . . . . . . . . . . | 93 |
| Footnotes | 96 |
| Bibliography | 100 |

List of Figures


Page

1. company/employee data structure diagram . . . 36
2. teacher/student/pupil data structure diagram 38
3. Part/product structure data structure diagram 39
4. husband/wife data structure diagram . . . . . 39
5. part/product structure data structure diagram with up and down names . . . . . . . . . . 42
6. Department/employee information in the network and relational frameworks . . . . . 49
7. Company/employee data structure diagram . . . 55
8. Comparison framework . . . . . . . . . . . . 60
9. Language Comparison Hierarchary . . . . . . . 63
10. Current language comparison status . . . . . 65
11. Part/Vendor/PO/Line-item data structure diagram . . . . . . . . . . . . . . . . . . . 67
12. Part/Product structure data structure diagram 70
13. Supplier/Project/supply data structure diagarm . . . . . . . . . . . . . . . . . . . 74
14. Supplier/Part/Project/Supply/Worker Data structure diagram . . . . . . . . . . . . . 77
15. Supplier/Part/Project/Supply/Worker Data structure diagram . . . . . . . . . . . . . 84

# Chapter 1 Introduction

## The Use of Computers

During the last quarter of a century, the use of computers has increased dramatically. Few will deny that computers have had a strong impact in many areas of our society, and anticipation of even greater future impact is widespread. Along with this growth, the nature of computer usage has changed since its inception, and we can predict further change in the near future.

The first computer users were scientists and engineers who needed greater numerical manipulation capabilities than was available at the time. The computational ability of these first machines, not their data handling capacity, attracted these communities, and, for the decade of the 1950's, they remained the primary users. The first widely used higher level language, FORTRAN, is clearly oriented to these computational needs; data handling in FORTRAN is primitive by comparison with currently available techniques. In fact the name itself, FORTRAN, is short for FORmula TRANslator, further indicating the emphasis.

With the reduction in cost of memory, and the development of secondary storage devices, the data management capabilities of computers came to be realized. With this, the business community became interested in the possibility

of using computers to handle the clerical aspects of their operations. Many simple, redundant operations could be performed cheaply and accurately on a computer, and the emphasis shifted from complex computations on little data to simple arithmetic operations on vast amounts of data. COBOL (COmmon Business Oriented Language), the next major higher level language to be introduced, reflected this shift from computation to data structure needs.

Up to this point, computers were considered only computational tools. People began to realize that, in certain structured problem areas, computers could make decisions. Any problem having a strict, procedural solution could be given to a computer, once that solution had been programmed, and the burden could be removed from the human decision maker. The area of inventory control is a prime example of such a structured decision being computerized.

Once computers were given decision making responsibilities, the natural extension was to expand the scope of decisions they could make into the area of unstructured problems. Before this could happen, a subtle but important shift had to occur in the view of computers. Previously, computers were thought of as systems unto themselves; man's interaction with them was of secondary importance. Then it was realized that the man and machine together could be viewed as a system which combined the effectiveness of the man

(intuition and ability to work with insufficient information) and the efficiency of the computer. Through this man/machine system or decision making unit, unstructured problems could be handled. The computer could provide data access and manipulation capabilities to allow the man to explore more alternatives in his search for a solution. The central idea was to extract as much structure as possible from a problem and program that, leaving the man to handle the problem of lack of structure. The program lets the man ask "What if ..." types of questions and provides simulated outcomes. The area of "decision support systems" is currently beginning to blossom.

Just as there has been an evolution in what computers are used for, there has been a parallel change in who is using them. As mentioned above, scientists and engineers were the first users. This was due not only to the computer's capability to fulfill their needs, but to the ability of these users to understand and use them. The first computers were formidable beasts and, thus, to use them required a technical background. Along the same lines, the transition into the business world was in part due to a "softening" of the computer interface. As decision support systems entered the arena, the user became higher level managers. To support this user, the interface had to shift more toward the human side. Much effort has been spent on natural language support,

graphics, and other techniques which would make the computer interface more palatable to the human user. The use and user of computers has evolved as a result of technological advances as well as effort to move the interface from the computer's side to the human user's side. Codd has predicted that the major user of the future (1990's) will be the casual user at home.[1] For this shift to occur, the interface must be pushed even further towards the human side.

## Current Problem Areas

Though the impact of computers has been great, many examples can be found of less than desirable results. One has to be careful not to be biased by the proliferation of disaster stories, for it is easy to be led to the conclusion that these dismal results are the norm. If this were true, the computer industry would not have enjoyed the success it has. How many businesses would continue to be burned if failure were the likely outcome? The problem is that disasters are easier to report (and to many more newsworthy) than successful applications. With this in mind, coupled with an intent to avoid the dramatic, we shall explore the problems that are most frequently experienced and seek to find their causes. An understanding of causes should help in the search for a solution.

From an end user's point of view, the quality of a software product is of primary importance. The problem one finds is not that software products are universally bad, but rather that there is great variability in the success of systems. Given the same task, one implementation may be much better than another, and some may never work at all. In many cases, especially in the current area of decision support systems, the software customer must accept a wide confidence region for his estimate of likely success. This makes software development a risky business, and, just as with a high risk stock, only a potentially high payoff will warrant the investment. Those customers who do not view substantial software development as a portfolio decision are the ones likely to be hit the hardest. The reputation of a vendor can go a long way to reduce this variability in expected success, but an even greater reduction could be hoped for.

The cost of software is usually the second concern of the customer, but in the current economic environment it is a close second. The benefit of using a computer can stem from one of two cases; either the computer can perform a task cheaper than it could be done other ways, or it can provide the ability to do things that could not otherwise be done. In the first case, the cost/benefit analysis has a basis for comparison, in the second the issues are usually too nebulous. In either case a cost reduction is desirable.

There are actually three costs associated with a software product, development cost, maintenance cost, and cost of use. Each of these has two sources, people and equipment. As it turns out, the cost of hardware has dropped two orders of magnitude over the last two decades, while the software cost necessary to utilize that hardware has continually been on the increase.[2] According to Madnick only 30% of the development and operation cost of new application software is accounted for by the hardware, and that ratio is still on the way down.[3] Thus the typically high cost of software is more a result of personnel costs than hardware costs, and therefore a reduction of personnel requirements could have the greater impact in terms of cost savings.

Businesses are typically dynamic, growing, evolving entities, and thus their software needs change over time. This being the case, software should be designed to allow for easy refinement. In many cases this is not done, and customers often find that starting from scratch is easier than modifying an existing package, even for seemingly simple changes. The requirement for flexibility is not only a function of time varying needs, but also one of initial specification inadequacies. Frequently a customer will not fully understand his own needs until he has had a chance to work with a system. If the system is built without flexibility for change, then by the time he realizes the

inadequacy of the initial specifications, it is too late.

One of the primary needs for software today is in the area of information storage and retrieval. Many businesses are coming to need database management systems to handle vast amounts of data, and the sheer size of these databases makes the problems discussed above even more pronounced. Because many consider database management the major bottleneck in software development, this will be the subject of this paper.

## The Reason for the Problems

The problems presented above derive from the fact that, given the tools (higher level languages) most commonly used to build software packages, it is a substantial jump to generate a finished system. Put another way, the difference between the finished product and the basic materials is quite large.

An analogy to housebuilding may clarify the meaning of the above assertion. In the early days of computers, a programmer had available only the primitive instructions of the machine with which to build programs. This is comparable to the good old days of the pioneers when a stack of felled trees represented the basic materials of a house. In each case there was a huge gap between the basic materials and the finished product. The next step for the programmer came

with the availability of higher level languages (FORTRAN, COBOL, PL/1, etc.). These languages afforded a base comparable to the use of precut boards in the building of a house. In each case, the gap from the materials to the finished product was reduced. While the housebuilding industry has advanced to the use of prefabricated units, higher level languages remain the primary tools of the majority of the software community. The reader desiring further support of this argument is highly recommended to refer to Simon's article, "The Architecture of Complexity."[4]

One can think of this gap in terms of driving a car from one point to another. If the distance is short, the number of alternate routes is small. If, on the other hand, the distance from the point of departure to the destination is large, there are many alternate routes from which to choose. A single route will likely consist of many legs or sub-routes.

A large gap from basic materials to finished system leads to the problems of the last section in the following ways.

* Obviously, the personnel costs depend on the time required to build a system. The more primitive the tools, the more time required. This can also lead to substantial lead times necessary to complete a system.

* The resulting design tends to be complex (many pieces with many interconnections). The complexity frequently eliminates the possibility of understanding the entire design at once. One must concentrate on only a portion of the design at a time, making it easy to overlook ramifications of decisions concerning that portion. Thus bugs are likely to creep into the design.

* The proliferation of potential paths makes any resulting design the personal choice of the implementer. How this choice is made will greatly affect the quality of the system (the variability of success issue).

* The fact that the path chosen is personal in nature ususally limits understanding of the design to the implementor. If he leaves, a change to the system can be all but impossible. This is why starting over is a frequent phenomenon.

## A Logical View of Data

The arguments presented above lead directly to the conclusion that the development of higher level tools is the solution. If the gap from basic materials to finished product is made small enough, the implementor could produce straightforward, easy to modify, working systems much faster

and at lower cost than ever before possible. To this end, we require an understanding of the common functions of all systems. Having restricted ourselves here to the area of database management systems, the framework necessary is a logical view of data or a theory of information. If we can develop a model of real world information, then that model would provide the basis or platform from which we could build information systems.

Over the years, numerous models of data have appeared. Some differ only in level of sophistication, one being a subset of another, while others represent different approaches. The next section shall highlight a few of the major models of the past and present. The subject of this paper will be a comparison of two of these views. It is important to emphasize here that comparisons of this nature can rarely be definitive, for the issues involved are highly subjective. One can merely make arguments for or against a model with respect to assumptions of what it is that makes a model good or bad. In this regard, one should always take care to explicitly state the assumptions underlying the argument, for it is likely that the assumptions will be the actual basis for agreement or disagreement.

The approach taken here to motivate the need for a logical theory of data is not the traditional one. We have discussed the framework in terms of providing tools to the user. The more typical strategy is to present a model as an interface which can clearly separate development efforts. In most cases the stated desire is to isolate the logical functions from the physical storage functions. Then, as new storage techniques are developed which could improve the efficiency of a system, the changes can be made without tampering with the logical functions (i.e., user application programs). This benefit of an interface is extremely important, but if it were the sole motivation, then just about any robust model would suffice. The current active research, despite the existence of several good canditate interfaces, tends to indicate that more is at stake. The model chosen should not only provide this protection from the physical structuring, but should represent the best platform from which to develop information systems.

History

The first big step on the road to a logical view of data came with the concept of a simple sequential file. System designers recognized that a typical data storage pattern involved storing many items of the same type. Thus the term record came into being to represent a single item, and a file

was a collection of these like records. From the computer's point of view, the record was simply a string of usually fixed size with no meaning; the interpretation was the responsibility of the user. Being able to think about data storage in terms of a file of records was quite an advance over having to format and use a secondary storage device with only its basic I/O commands. Commands to read and write logical records from a file were provided, and the gap from basic materials to finished product was significantly reduced.

Soon demands became too complex to be handled by a simple sequential file, and random accessing techniques were developed. It was recognized that a user would like to access records by some meaningful name as opposed to a logical record number, so indexing and hash-coding techniques arrived to handle the need efficiently. Even with the added sophistication, the underlying model of data, a single collection of like records, remained the same. This model, though fairly simple, has enjoyed great success as attributed by the fact that it is still the most commonly used model of the software community.

To this point, the contents of records were meaningless to the computer system. Soon ideas developed concerning standard strategies for interpreting the contents of records and relationships between them. The need for a theory of information which would support the ability to interpret the

contents of records was soon felt. Over the years quite a few models have appeared in the literature. The undercurrents of most of these models fall into one of two general catergories, network or relational. One finds that this dichotomy clearly delineates two camps, and there is an active debate in the literature between them.

The CODASYL Data Base Task Group has been the leader in advocating the network model. In 1962 they began the task of developing, along with a standard model, a proposal for a standard Data Definition Language (DDL) and Data Manipulation Language (DML). They are still actively working toward that end. Along the way, many successful systems have been built on the network model.

In 1969, E.F. Codd of IBM Research in San Jose began advocating the relational model.[5] Its underlying goals are similar to those of the network model, but the approach is different.

We have not explicitly mentioned the hierarchical view of data, despite its importance as a basis for some widely used data management facilities (such as IBM's IMS). The hierarchical model can be viewed as a subset of the network model and has been found to be inadequate in representing many real world information structures. For these reasons, we will not consider the hierarchical model, even though it does represent substantial current usage.

The Current Conflict

As mentioned above, the debate between networks and relational advocates is currently quite active. This paper will attempt to provide a framework for the comparison of the two models, hopefully putting many of the current arguments into perspective.

Because any comparison of this type is highly subjective, there are many potential pitfalls to be avoided. A description of several of these follows:

* Because there is no single universally accepted definition of each model, there is a high degree of latitute in what one assumes to be the components of each. There is no clear resolution to this problem, and the assumptions one begins with are likely to have more impact on the acceptance of the results than the process of reaching them. This paper attempts to deal with this problem in two ways. First, we begin with the author's reading of the accepted bases of the two models. The determination of the accuracy of this reading is left to the reader. Second, we do not penalize either model with this reading. If either model is found to be deficient, and the author finds a way to remedy the problem without altering the underlying structure of the model, then a change is not

ruled out. Essentially we are giving each model the benefit of the doubt, and using the latitude mentioned above, hopefully, to make the comparison more meaningful.

* Due to the greater age of the network model, more systems have been built on it. A network advocate could argue that the greater number of systems built on the network model is an indication of its greater suitability. If one model inherently leads to a more efficient implementation, that might be a valid point of contention, but enough time has not passed to determine that. From the other side of the coin, the relational advocate has more targets for his attacks on the network model. It is easy to equate problems in a particular implementation with problems in the model underlying the implementation. This should be avoided at all cost, and any use of an implementation as a focal point for comparison must be accompanied by a clear analysis separating the problems due to the model from those due to the implementation. By and large, this paper avoids specific implementations as bases for comparison of the two models.

* The relational model revolves around an abstract mathematical theory, whereas it is difficult to find precisely stated mathematical descriptions of the network model. This leads the network advocate to argue that practical issues have not been considered in the relational model, and the relational advocate to argue that no mathematical basis for the network model exists. These arguments are meaningless. The relational argument will be countered, to some extent, by a proposal for a network theory in terms of the relational model. The author claims minimal abstract mathematical background, and hence the presentation will be non-rigorous. It is hoped that someone with a stronger mathematical bent will pursue this course further. It might be suggested, as an aside, that the reason for the absence of a mathematical formulation of the network theory may be due to a lack of felt need, the possible result of successful implementations.

* The network model of data leads directly to a simple implementation strategy. It is quite possible that the model actually grew out of this strategy. For the relational model, on the other hand, the most straightforward implementation strategy would be horribly inefficient. This could be offered as an advantage of the network model, but it is not an important point

because the implementation is done, hopefully, only once. It is interesting to note that this same fact has been used by relational advocates to argue that the network model is not a logical model of information but rather is a technique for organizing the storage of physical records. This paper will present the network model as a logical model of real world information, and the fact that it leads to straightforward implementation will not be counted against it.

## Presentation Strategy

The reader may detect a slant in this paper in favor of the network view. Though the author does not deny a certain tendency in that direction, the purpose of the paper is to present a more objective framework for comparison. In each point of comparison, an attempt was made to present the arguments as fairly and with as little bias as possible. The goal has been to find the better of the two models, if possible, and not to merely defend one model on an emotional basis. The slant is intended, in part, to overcome any subjective endoctrination the reader may have absorbed from the relational literature.

The paper can be divided into four parts.

1. Background (Chapter 2 and 3)
   The network and relational models will be presented to the extent necessary for our purposes. Both the general reading and the author's modifications will be discussed.
2. Comparison (Chapter 4)
   Five points of view with respect to a database management system will be presented with a comparison of the two models in terms of each.
3. Mathematical development of the Network Theory (Chapter 5 and 6)
   To put the network model into a more theoretical framework than it has had in the past, a network algebra and a network calculus will be developed (a la Codd).
4. Summary (Chapter 7)
   A recap of the major results will be presented. The supportability of each view by the other model will be discussed. The potential benefits of a hybrid view will also be given. Finally, topics for further research which have been generated here will be itemized.

# Chapter 2 The Network Model of Data

## Introduction

As mentioned before, many database management systems have been built on the network model of data. Looking, however, at some of these individual systems, it is not clear that there is a single model underlying them all. Each designer seems to have his own interpretation or variation of the central theme of the network model, and this makes difficult the task of explaining "the" network model. The model set forth here represents the author's understanding of the central concepts of the network model, and thus may be at odds with other expositions. The bulk of the ideas and terms are those presented by Bachman and his articles should be referenced for further clarifications.[6,7,8]

## Data versus Information

Before plunging into a discussion of the terminology and concepts of the network model, it may be profitable to clarify the difference between the terms data and information. A data element is simply a value; for example, the number 27. By itself it has no meaning. Information is interpreted data, or an association. If 27 is the number of people in a particular class, then the data element 27 has taken on meaning, and hence informational content, by virtue of its interpretation.

The function of a database management system is not just the storage of vast amounts of data as the name might imply; it is rather the storage of information. Database management systems revolve around the concept of an interpretation of the data they store.

## Entities, Attributes, and Keys

All information exists with respect to objects or things which shall be termed entities. Any concrete or abstract object can be an entity. For example, a person, a state, a color, and an idea are all entities. In a database management system, entities are the things about which we wish to store information. We shall call the information we wish to store about an entity its attributes, and the names we give each attribute will represent the interpretation of the associated data. For instance, we may wish to remember that the age of Tom Smith is 32. Then the value 32 is the age attribute of entity Tom Smith. In fact, "Tom Smith" itself is an attribute, the name attribute of a particular person entity.

The name attribute, as we have been using it, plays a special role. We used the name Tom Smith to actually mean the entity itself. Whenever an attribute or a group of attributes uniquely identifies an entity, that attribute or group of attributes is called a key. Every entity represented in a database management system must have a key, even if it is

composed of all the attributes of the entity. Likewise, any entity may have many keys. If this is the case, one is arbitrarily chosen and deemed the primary key for use by the system.

Entity Classes

As mentioned above, any concrete or abstract object can be an entity. Two examples are the person Tom Smith and the color red. Even though these are both entities, it is useful to distinguish between them because the sets of attributes describing them are different. Tom Smith, for example, would not have the attribute wavelength, just as the color red would not have a social security number. We thus define an entity class to consist of all entities described by the same set of attributes. There might be a person entity class with the attributes name, age, and social security number, and a color entity class with attributes color name and wavelength. We thus have a scheme to classify all entities according to the attributes that describe them. In some cases, the classification of entities may be a matter of judgement, depending on circumstances. For example, men and women are largely described by the same attributes. In one system it may be more profitable to have one entity class for both, along with an attribute sex. In another, it may be preferable to define an entity class for each. A school database would most likely combine men

and women into a single class whereas a medical database may benefit by splitting them. The trade-offs of the application will determine the appropriate dichotomy.

## Associations Between Entities

To this point we have described a database to consist of a group of unrelated entity classes. Each entity class is separate and maintains the values of the various attributes associated with the entities in a class. The one additional type of information we might want to record is the association between entities. Some typical examples are the associations between:

* husband and wife
* teacher and student
* father and son
* company and employee
* assembly part and component part

Thus, instead of associations between an entity and an interpreted value, each side of the association is an entity. It is the handling of this type of association that clearly differentiates the network from the relational model. Any other characteristics of one model, if beneficial, could easily be incorporated into the other. It is, therefore, in this

area only that valid comparisons can be made.

Before describing the network approach to managing associations between entities, let us explore the characteristics of such associations. As a first step, what is the number of entities which may participate in an association? In each of the examples given there were only two entities, but this may not always be the case. For example, we may wish to associate a mother, father, and child. Thus an association may be of any degree, or there may be any number of entities taking part in it.

In some cases, as in the company/employee association, the role of an entity in an association will be clearly understood based on its entity class. In other cases, as in the father/son association, two or more entities in an association will come from the same entity class. The role of an entity is then not clear. If Tom Smith and John Smith are associated in the father/son association, it is unclear which is the father and which is the son. Thus, for each association, we will assign role names to each part played in the association. We can thus think of a single association as relating two or more entities, each in a certain role with respect to the association, with no restriction that the various entities come from distinct entity classes. Note that we have not ruled out the possibility of two or more roles being the same, as in the case of associations between brothers.

When an association has two identical roles, they are said to be symmetric; if A is the brother of B, then B is the brother of A.

We have been treating an association as a single instance of an association between specific entities. It now becomes beneficial to differentiate between an instance of an association and as association class comprising all instances of a particular association type. For example, all instances of associations between fathers and sons comprise the father/son association class, just as all instances of person entities made up the person entity class.

Now, considering only binary association classes for the moment, how many instances of an association class may have the same entity in one role. In the example of a husband/wife association, assuming a monogamous society, a single man may only appear in one instance of the association. Clearly the same is true of a women. Thus only one instance of an association may have a single man entity in the husband role or a single woman entity in the wife role. Put another way, a single man may take part in only one instance of the association, and the same is true for a single woman. This association is thus termed one to one (1:1). Now, in the company/employee association (assuming a person may only work for one company) one company may occur in many instances of the association, but each employee may occur in only one

instance. This is called a one to many (1:n) association. Moving on to the teacher/student case, one teacher may have many students and hence be found in many instances of the association. Likewise, one student can have many teachers and hence be found in many instances of the association. This is a many to many (m:n) association. Thus binary associations may be one to one, one to many, or many to many.

The network model of data, as expressed in the literature, provides directly for the storage of one to many binary associations only. This is actually not as restrictive as it sounds. Take first the association degree issue. A set of binary associations can always represent an association of higher degree. For example, in the father/mother/child case, two binary associations, father/child and mother/child, serve the same purpose. By operations on these two associations, one can derive the associations between mothers and fathers. As far as many to many associations are concerned, they too can be handled via the creation of a new entity class. A one to many association is defined between each of the two entity classes to be (m:n) associated, and the new entity class via operations to be defined shortly. This "cross reference" entity class provides the needed many to many association capability. Unfortunately, there is no way to restrict a one to many association to handle the one to one case. In one sense, a one to one association can be considered a subset of

the one to many case, where many in each instance is one. But the object would be to have the system enforce the one to one restriction, not the user. The author contends that the omission of one to one associations results in an incomplete data model and hence its inclusion will be assumed for the purposes of this paper. In summary, a data model which provides one to one and one to many binary associations possesses the basic capabilities necessary to handle all meaningful real world associations.

Many may consider the omission of a direct facility to handle many to many relations to result in an incomplete model, and the use of a cross reference entity class to be an artificial solution. There is, however, a way to look at many to many associations which may help to overcome this concern. Consider the student/teacher example. Any student is not associated with a "whole" teacher, but rather with a part of that teacher's day, the class period. Thus the cross reference entity class, which to this point has been described as an artificial means to handle many to many associations given explicitly only a facility to handle one to many associations, may take on a real, logical meaning unto itself. Thus the cross reference entity between teacher and student entities, would represent the pupil/class entity. Each pupil/class entity is associated with one student and one teacher. Any student may be associated with many pupil/class

entities, as may any teacher. Any time there is a many to many association, there is an underlying implication that one of the entities on either side cannot be associated with "all of" any entity on the other side.

The one to many binary association can be thought of as directed because there is an asymmetry between the two roles of the association. One role can have any entity only once in all instances of the association class, whereas the other role may have the same entity appearing in many instances. To differentiate the two roles, the role in which an entity may appear only once is called the member of the association and the other is called the owner. For example, the company would play the role of owner with respect to the association with member employees. These terms can apply equally to the entity classes related by the associations (if distinct), or to individual entities which constitute an instance of the association. The reader should take care that conotations of these terms do not interfere with the analysis of database structure. The fact that companies do not "own" individuals does not mean that their database counterparts should not. Owner and member are purely technical terms to differentiate the two roles in a one to many binary association.

The term key has a special meaning in connection with associations in the network model, and we must revise our former definition slightly. As used previously, keys served

to uniquely identify entities in an entity class. As commonly used in the network model, a key is an attribute which orders and identifies members of an association. Thus each association is identified with a key or attribute in its member entity class, and all database accesses are made by following associations to the desired entity, using the keys to direct the search. This access pattern in terms of paths through associations tends to differentiate between two types of entity classes, those which are directly accessible and those which must be accessed as members of other entity classes. Some systems manage these two types in the same fashion by creating a special root entity. Each directly accessible entity class is the member of an association with the root entity. Thus all accesses in the database begin at the root entity and follow a path through entity classes via associations. In this scheme, it is clear that the set of keys in a path uniquely identifies the entity at the end of the path. Thus this revised definition of keys as identifiers within an association class serves the same function ascribed to them previously, uniquely identifying entities in an entity class. This combination of functions, however, does cause problems. If a particular entity can be accessed via two distinct paths, then there can be redundant unique identifications of an entity. Sometimes one or more of the keys is allowed to be non-unique to handle this problem. This

also can lead to problems in creating and deleting using the non-unique path. The author feels that the functions of ordering members and uniquely identifying entities should be clearly divided. It is suggested that the term "key of an association" be used to mean the attribute(s) which orders the members of the association. Unique identification of an entity in an entity class should then be expressed in terms of the set of attributes and associations which guarantee uniqueness. This set will be called the identification key or primary key of the entity class. Unfortunately, the word "key" has developed in the literature to handle both issues, but the appropriate qualification or context should make clear the function intended. The reader should keep clear in his own mind the two uses of the term and be careful to understand which is being used at any point.

What are the basic functions required to manipulate associations? Given any owner entity with respect to an association, we should be able to find any or all members. In most systems, when requiring all members, there is usually a "piped" facility to present the members one at a time in order of the key of the association. Given the owner and a member, this facility allows us to retrieve the previous member or the next member. Also, given any member entity of an association, we should be able to find the owner.

## Standard Terminology

In the network model of a database management system, there is a file for each entity class. Each record in the file corresponds to one entity of the class. The attributes of entities are stored in fields on each record. The associations between entity classes are directed relations between the owner and member files.

## Data Structure Diagrams

A data structure diagram is a technique for depicting graphically the files and relational structure of a database The technique is quite simple, using only two symbols, a box and an arrow. A box represents a file or entity class. There is one box for each file in the database. An arrow represents a relation between two files, pointing from the owner file to the member file. With these simple tools we can represent any complex database structure.

For example, the company/employee relation discussed above would be depicted:

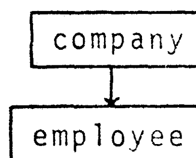



Figure 1

The relation (arrow) is frequently thought of as a chain. Given any company we can go down the chain to find a particular employee; we can find the previous or next employee in the chain for this company or we can find the company.

It is common practice, when possible, to place owner files vertically above their corresponding member files.

## Trees and Networks

Using data structure diagrams, we can explore the implications of various database structures.

In the company/employee database, each student had only one owner entity or only one relation in which it played the member role. When each file in a database is a member of at most relation, that database is said to have a simple tree structure. Simple tree structures are frequently inadequate to describe a real world information structure. To fit the teacher/student database into a tree structure, we would have to make the assumption that each student has only one teacher. This is not commonly the case.

A database has a network structure when any file is owned by more than one relation. The teacher/student database will illustrate a network. During a school day, a student has several classes, usually each with a different teacher. To model this information structure, the following diagram is adequate.

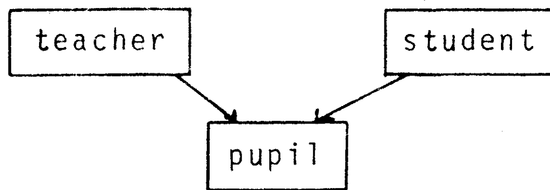



Figure 2

Each record in the pupil file represents a class period for a particular student. Thus the fields of the pupil file might be class name, period number, and class room. Given any student, we could chain through the pupil file, finding the associated teacher (owner) record. Likewise, given any teacher, we could chain through the pupil file, finding the associated student.

All the structure discussed to this point has existed in the structure of the files. In a special network, it is possible to store structure in the data itself. For example, in a manufacturing environment there might exist a part file. Each part is either a detail part or is made up of other parts (it is an assembly). The parts of an assembly might be assemblies themselves. As the product structure varies, new files should not have to be created or deleted to maintain that structure. All we really need beside the part file is a file to record each instance of a use of one part in the assembly of another. Each record in this product strucutre file is related to two part records, the assembly part and

the component part. Each part record may have many components or be used in many assemblies. The following file structure brings this together.

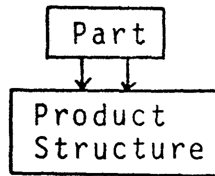



Figure 3

Given any assembly part, we can travel down the component chain. At each component record we find its owner via the other chain (the component part record) and check it for components. This double chaining between two files allows us to store a network structure in the data itself.

To diagram the one to one relation, which we said would be considered part of the network model for our purposes, we need use only an arc instead of an arrow (directed arc). The husband/wife database might appear as below.

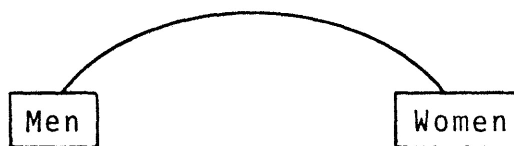



Figure 4

The use of an arrow to indicate the direction of a one to many relation has been, on occasion, a source of confusion, for frequently it will be thought of as a pointer. This results in the misconception that it is only possible to go from an owner to its members via the relation. In the discussion of basic functions we noted that one requirement was to be able to find the owner of an association given any member. Thus is a sense we can traverse the arrow in either direction; the arrow merely distinguishes the two sides of the relation.

Relation Naming

In most of the diagrams given, we have not explicitly labelled the relations. This does not imply, however, that the naming of relations is irrelevant. In many cases it is crucial. In fact, as we shall see, a relation may actually need two names, depending on the direction travelled. In the case where there is a single relation connecting two entity classes, the need for explicit naming is reduced, for then the ambiguity question does not arise. In a strict hierarchy, relation naming can be ignored.

Consider the part/product structure database depicted in Figure 3 . Given any part record, we may want to find all components used to make that part or all the assemblies that part is used in. Which task we want to accomplish will

determine the appropriate path to follow. To communicate the request to the system, each path should be given a name which denotes its function. In this example, two likely names would be "component" and "assembly". For any part we would find all members of the component relation to get the components used in the part, or all members of the assembly relation to find the assemblies the part is used in. Now let us consider that we have a certain product structure record and we want to find the assembly part number it corresponds to. Unfortunately, it is the component relation that connects an assembly part to its component product structure records. Thus, to get the assembly part number of a product structure we must find the owner via the component relation. The thrust of this argument is that a name which is meaningful for a relation when looked at from one side may not be meaningful when looked at from the other. The implication is that a relation should have two names, one to be used when looking "down" the relation from the owner side, and one to be used when looking "up" the relation from the member side. The naming for the part/product structure case might be:

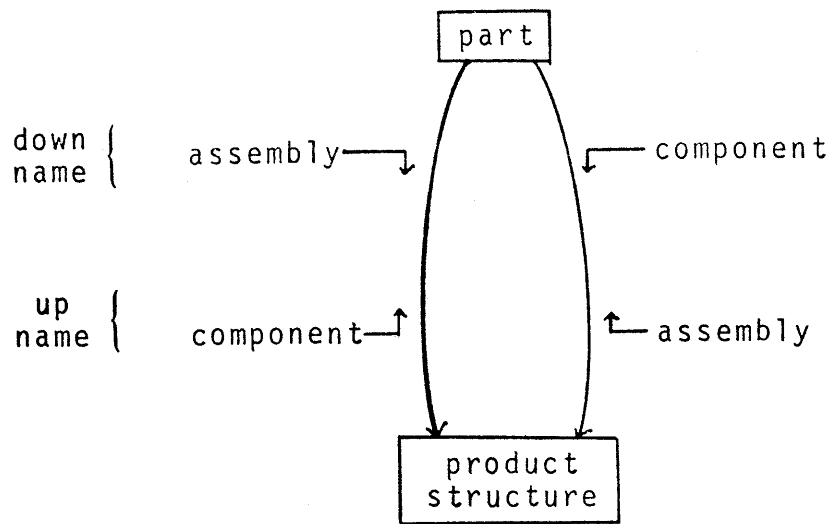



Figure 5

# Chapter 3 The Relational Model of Data

## Introduction

The relational model of data is based on mathematical set theory of relations. The application to information theory has been the effort of many, but E.F. Codd of IBM research in San Jose, California, is by far the leading (and most prolific) proponent. The description of the relational model given here is almost exclusively based on the work of Codd.[9,10,11,12]

The term "relation", unfortunately, has an entirely different meaning in the relational model than in the network model. This is a continual source of confusion in attempts to discuss or compare the two theories, but each use of the term is so firmly and centrally rooted in each theory that one must simply be careful to interpret it in terms of the theory to which it applies.

## Goals

The relational theory was developed to overcome three problems noted in many systems. Codd firmly asserts that the relational model is not a response to the network model, but the first major paper discussess the problems in terms of the shortcomings of the network model, among others.[13] Whether these three problems were inherent in the network

theory or merely outcomes of particular implementations is a subject for later discussion. The three problems are those of ordering dependence, indexing dependence, and access path dependence.

Ordering dependence means that files and relations in the network model are stored in a particular order of the keys (usually the alphanumeric collating sequence). For instance, given a company record and requesting all the employees of the company, a system would usually be designed to present them in a predetermined sequence, for instance in order by employee number. It was argued that application programs would be written expecting that order (depending on it) and thus, if the stored order were changed over time as may well be the case, the application program would no longer function properly. The reason relations are ordered in almost all network based systems is one of efficiency. First, the user will almost always request the same order for a particular relation. This has been observed in practice. Second, there are certain system operations which can be performed more efficiently if ordering exists in a relation. As an aside, perhaps application programs should specify the order they require when requesting members of a particular relation. If the order requested matches the system ordering, the specification may be ignored; otherwise the system can be expected to reorder (sort) the members prior to presentation.

Thus ordering can represent a potential cost savings but its possible consequences should be understood and planned for. To return to the mainstream of the discussion, ordering is dispensed with in the relational model so that application programs may not be designed which exhibit ordering dependencies.

Indexing dependence is purely an efficiency issue in implementation and has no bearing on a logical view of data. Indexing is a technique for rapidly finding a particular record given its key. The logical function is to find a record given its key. Whether this is done via indexing or by a logically equivalent linear search should be irrelevant to the application program. Unfortunately, as Codd points out, some systems such as IDS require the application program to reference an index by name rather than have the system check for its existence and use it.[14] Indexing has no place in the logical network model and any implementation in which indexing is not transparent to the application program cannot be said to support the logical network model.

Access path dependence is a truly inherent aspect of the network model, but its ramifications are not quite those generally expressed in the literature. There are two ways to look at the network model, as a logical model of real world information, and as an implementation strategy. Those that argue against the network model in terms of access path depen-

dence usually treat it as an implementation strategy. Thus they present several possible structurings in which to store the data and thus demonstrate that an application program will depend on which structuring is chosen. If the structure is changed, the program will no longer function properly. Treating the network model as a logical view of information, the problems are always diminished if not altogether removed. If the logical view is followed, there is one and only one structure which accurately reflects the real world relationships between entities. Thus if the real world structure is constant, so is the network model, and the concern over switching possible structurings is banished. If, on the other hand, the structure of the real world changes, then the database structure must be altered to reflect the change if it is to remain an accurate representation of the world. In this case, the only meaningful one, application programs may, in fact, require modifications. Thus the relational model is proposed as a scheme to maintain the invariance of application programs as the database evolves over time.

## The Model

The relational model is based upon abstract set theory. It assumes a collection of pools of values called domains. For example the sets of all possible names or colors or ages or part numbers are all domains. A relation on domains $D_1$,

$D_2$, $D_3$, ... , $D_n$ is defined to be a set of n-tuples, each of whose first element comes from domain $D_1$, second element comes from domain $D_2$, ... and $n^{th}$ element comes from $D_n$. A relation, then, is the counterpart of a file in the network model; each domain corresponds to an attribute, and each n-tuple corresponds to a record. Each domain is usually given a name to identify it so that the ordering of domains is no longer necessary. If the same domain appears more than once in a relation, each is given a role name to distinguish it from the others. At any point in time, the term active domain refers to the subset of a domain which actually appears in relations in the database. All tuples in a relation are distinct. A subset of domains in a relation which uniquely identifies a tuple is called a candidate key. A candidate key is non-redundant if none of the domains in the key are superfluous in uniquely identifying a tuple. If there is more than one non-redundant candidate key, one is chosen and termed the primary key of the relation. If a domain or set of domains in a relation acts as the primary key in another relation, it is called a foreign key.

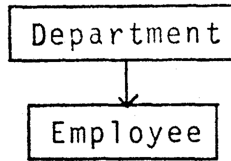
## First Normal Form

Up to this point we have made no restriction on the domains in a relation. The elements of a domain may in fact be relations themselves. For example, an employee may have

as a domain his salary history. This could itself be a relation on the domains date (of hiring), date (of termination), and salary. Any domain which is itself a relation is termed a non-simple domain. A relation is said to be in first normal form if it is free of non-simple domains.

Codd has defined a normalization process to reduce a relation containing non-simple domains to a set of relations in first normal form.[15] If relation R has a non-simple domain a, then a relation A is created. For each tuple s in the domain a of each tuple r in relation R, a tuple t is put in A which is made up of the primary key of r and the domains of s. Once A is created, the non-simple domain a is removed from R. Iterations of this process until no non-simple domains remain results in a database in first normal form. Viewed in terms of the data structure diagrams of the network model, the attributes of the primary key of any owner file are merged with the attributes of each of its member files, and the relations are removed. For example, consider a relation between a department and the employees who work in the department.

Network




| File | Attributes |
|---|---|
| Department | Department # (key)<br>Department manager #<br>Department name |
| Employee | Employee # (key)<br>Employee name<br>Employee age |

| First Normal Relation | Domains |
|---|---|
| Department | Department # (key)<br>Department manager #<br>Department name |
| Employee | Department # (key)<br>Employee # (key)<br>Employee name<br>Employee age |

Figure 6

Operations on Relations

Frequently it is necessary to perform operations on a relation or a set of relations to extract the information required by a query. These operations can be broken down into those which operate on a single relation and those which operate on more than one relation. Some are commonly found in set theory and others are specifically devised for the needs of a database system. The description of each below is primarily taken from Codd.[16,17]

I. Single Relation Operations

1. Projection

This is commonly an extraction of certain domains from a relation. It provides also for reordering (permutation) of domains in a relation and a repetition of domains. If A is a list of integers, each between 1 and the degree of a relation R, and r is a tuple from R, then r[A] is a tuple whose $i^{th}$ domain is the $j^{th}$ domain of r where j is the $i^{th}$ element of A. Each tuple of R is used, under the list A, to generate R[A]. Remember that a relation may not have identical tuples, so projection may require the removal of redundant tuples from the resulting relation.

2. Restriction

This is a means of selectively removing tuples from a relation based on a given test function. The allowable test functions are the standard value comparison tests $<, \leq, =, \neq, \geq, >$. A restriction on R between domains A and B based on the test $\theta$ can be expressed:

$R[A \; \theta \; B] = \{r : r \varepsilon R \wedge (r[A] \; \theta \; r[B])\}$

Of course, A and B must be comparable types of values, i.e., both numbers or both character strings.

II. Multi-Relation Operations

1. Union Intersection Difference

These standard set operations apply only to union compatible relations, or relations defined on the same domains. They are defined in the standard way.

$R \bigcup S \equiv \{r : (r \varepsilon R \vee r \varepsilon S)\}$

$R \bigcap S \equiv \{r : (r \varepsilon R \wedge r \varepsilon S)\}$

$R - S \equiv \{r : (r \varepsilon R \wedge r \not\varepsilon S)\}$

2. Cartesian Product

This is also a standard set theory definition.

$R \otimes S \equiv \{r^\frown s : (r \varepsilon R \wedge s \varepsilon S)\}$

($r^\frown s$ means the concatenation of tuple r with tuple s)

3. Join

If $\theta$ is one of the comparison operators $<, \leq, =, \neq, \geq, >$, then the $\theta$-join of two relations R and S on the domains A (in R) and B (in s) is defined.

$$R[A \ \theta \ B]S = \{(r^\frown s) : r \varepsilon R \ \wedge$$
$$s \varepsilon S \ \wedge \ (r[A] \ \theta \ s[B])\}$$

The most common use of a join occurs when $\theta$ is the equals operation. This equi-join results in two identical domains in the resulting relation. If one is removed, the result is termed the natural join. It is clear from the definition that the join is not a basic operation because it is expressible in terms of the cartesian product and restriction.

4. Division

This is an inverse operation to the cartesian product since

$$(R \otimes S) \div S = R$$

Division is an extremely complex operation to understand, and because it will be of important use later, it is suggested that the reader become thoroughly familiar with it before proceeding. Before giving a formal definition, we will describe it in terms as close as possible to the use it will serve later. Assume T is a relation which can be partitioned into two sets of domains such that the second set is union

compatible with a relation S. R, a relation we are seeking to derive will have the same domains as the first set of domains of T. R will consist of all tuples r such that, for all tuples s of S, $r^\frown s$ appears in T. If R is thus derived, then

$$T = R \otimes S \cup W$$

where W can be thought of as the remainder. It is impossible to express any subset of W as $X \otimes S$. For the formal definition,

"Suppose T is a binary relation. The image set of x under T is defined by

$$g_T(x) = \{y:(x,y)\varepsilon T\}.$$

Consider the question of dividing a relation R of degree m by a relation S of degree n. Let A be a domain identifying list (without repetitions) for R, and let $\bar{A}$ denote the domain-identifying list that is complementary to A and in ascending order. For example, if the degree m of R were 5 and A = (2,5), then $\bar{A}$ = (1,3,4). We treat the dividend R as if it were a binary relation with the (possibly compound) domains A, $\bar{A}$ in that order. Accordingly, given any tuple $r \varepsilon R$, we can speak of the image set $g_R(r[A])$, and we note that this is a subset of R[A].

"Providing R[A] and S[B] are union-compatible, the division of R on A by S on B is defined by

$R[A \div B]S = \{r[\bar{A}]: r \in R \wedge S[B] \leq g_R(r[\bar{A}])\}$.
Note that, when R is empty, R divided by S is empty, even if S is also empty."[18]

Division is not a primitive operation in that it is describable in terms of the projection, cartesian product, and difference operations.

$R[C \div D]S = R[\bar{C}] - ((R[\bar{C}] \otimes S[D]) - R) [\bar{C}]$[19]

## Second and Third Normal Form

A formal description of and motivation for second and third normal form can be found in Codd's articles, "Further Normalization of the Relational Database Model," and "Normalized Data Base Structure: A Brief Tutorial."[20,21] We shall content ourselves here with a description of second and third normal form in terms of the network model.

As described previously, the process of reducing a network data structure diagram to a set of relations in first normal form consisted of merging the primary key of any owner record with all of its member records. If then the (network) relations are removed, a set of (relational) relations in first normal form are produced. We have yet to specify whether these inferred primary keys of the owner file take part in the primary key of the relation derived from the member file. The answer is that, if the association or relation between the two files was a part of the primary key of the member

file, then the merged primary key from the owner will become part of the primary key of the relation derived from the member file. Consider the company/employee network diagram.

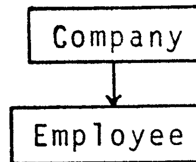



Figure 7

Assume company number is the primary key of the company file. Now there are two possible cases for the primary key of the employee file. If an employee number is unique (i.e., social security number is used), then the primary key of the employee file is simply the employee number. If on the other hand employee number is only unique within a company, the employee number <u>and</u> the company/employee relation constitute the primary key of the employee file. If this network structure were reduced (normalized) to the relational model, then, in the case of the universally unique employee number, the primary key of the employee relation would only be the employee number. If, on the other hand, the company/employee relation was also part of the employee file primary key, then the resulting employee relation would need both company number and employee number to make up its primary key.

What does this have to do with second and third normal form? The problems to be overcome by both second and third normal form can be viewed as the result of an error in normalizing an accurate (in terms of the real world) network structure. The error is to merge into the member file more than just the primary key of the owner. For example, if company location as well as company number were normalized to the employee file, then the resulting relation would not be in either second or third normal form.

If the cause of a relation's not being in second normal form is the same as the cause of a relation's not being in third normal form, what is it that distinguishes the two? The only distinction between the two depends on whether or not the normalization error occurred on a (network) relation which is part of the primary key of the member file. If the relation of the error takes part in the primary key (or more precisely any candidate key) of the member file, then the resulting relation is not in second normal form. If the relation is not part of a primary key, then the relation will be in second normal form (assuming no other errors) but it will not be in third normal form. Viewed with this perspective, the distinction between second and third normal form is fairly trivial.

Algebra and Calculus

Codd calls the operations on relations described earlier a relational algebra.[22] Queries formulated in terms of this algebra are, in essence, specifying a procedure for the extraction of desired information. He has also developed what he terms a relational calculus which provides a language in which queries can be formulated in descriptive rather than procedural terms.[23] The relational calculus represents Codd's proposal of the proper interface between user queries and the system, a high level, precise, description language. To illustrate that this relational calculus is "relationally complete" (can be formulated in terms of the relational algebra), Codd defines a reduction algorithm to take statements in the relational calculus and reduce them to statements in the relational algebra.[24] Thus he demonstrates that any query which can be formulated in the relational calculus can also be formulated in the relational algebra.

We will not pursue the details of the relational calculus and reduction algorithm here. The interested reader is referred to Codd's paper, "Relational Completeness of Data Base Sublanguages."[25] Altered versions of these will be found in the chapter on the network calculus. Those interested in the success of the relational calculus as a target language for interpreting user queries are referred to

Codd's paper, "Seven Steps to Rendezvous with the Casual User."[26]

# Chapter 4 Viewpoints for Comparison

## Introduction

When comparing two models, it is important to clarify at the outset the point of view from which the comparison will proceed. Aspects which may be useful or beneficial with respect to one point of view may be disasterous from another. The goals of the viewpoint will be crucial in sorting out the various issues. To insure that this procedure is followed, we will begin with explicitly stated points of view and analyze how its goals are satisfied or not satisfied by each model. Another approach would be to itemize characteristics of each model and analyze them in terms of who or what is aided or hurt by each. This approach, however, leads to difficulty in final evaluation and it is often hard to ascertain which characteristics will have important or meaningful impacts. With the former approach, we can come to more conclusive results in each viewpoint.

The following diagram will establish a framework for comparison of the two models. All relevant entities which could be effected by the model choice are illustrated and they will be dealt with one at a time.

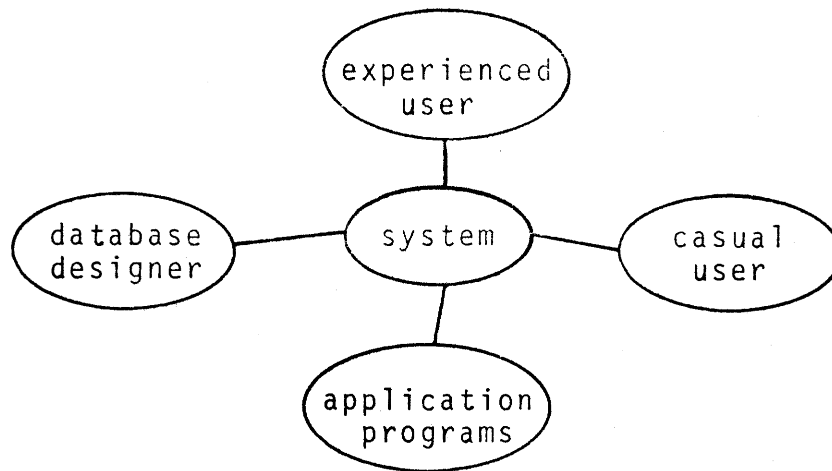



Figure 8

Database Administrator/Designer

The database designer is the one responsible for determining what information is to be stored in a database, and how that information is to be organized. It is extremely important that the designer insure that the organization is a true representation of the real world.

In the relational framework a true representation of the real world is equivalent to having relations in third normal form.[27] As discussed in Chapter 3, this means that all functional and transitive dependencies have been removed. So, from the system designer's point of view, what is the process involved in removing these dependencies?

Codd provides a mathematical formulation to express what each of these dependencies looks like.[28] With this, then, can the system purge itself automatically of these dependencies? The answer is no because knowledge of the real world

counterparts is required. Hence the designer must interact with the system to remove them. The best a system can do to lead the designer through the process is to ask, through a systematic series of questions, whether each potential dependency in the current database formulation is, in fact, a true dependency. If so the removal is rather simple. Grant Smith has actually devised a strategy for asking the questions.[29] In a database of any complexity, the number of potential dependencies is huge and hence, the question and answer session is likely to be a tedious one.

How do these functional and transitive dependencies, errors in the representation of the real world, ever make their ways into the relational organization of the database as formulated by the designer? The answer is inherent in the framework of the relational model. Dependencies are the result of storing an attribute of an entity in a relation that does not represent that entity. In the relational model, each relation is considered to be a table distinct of all other relations in the database, which stores associations that the userwishes to keep track of. Thus the user is led to think about only one relation at a time. Since he probably designs it thinking in terms of some desired application or report, it is a natural process to include in it all the information required by the application or report. The disjoint nature of a relational database, then is the

cause of the introduction of functional and transitive dependencies. The user is encouraged not to think about associations between relations.

In the network model, the designer is forced from the outset to consider the relationships between files. Before he can begin storing attributes he must structure the database to represent the relationships between entities in the real world. With the overall picture in mind, he decides which attributes are needed and where they should be stored. It is difficult indeed to store an attribute in the wrong file, the error almost stares you in the face. Although a series of questions could easily be devised to check the validity of each attribute, it is unnecessary because mistakes are rare or nonexistent.

Thus insuring the consistency of a database structure is a natural outcome of utilizing the network model, whereas it entails a tedious check-up job in the relational scheme. It is the inherent psychology underlying each approach that makes the relational model more prone to this type of error than the network model.

## A Framework for Language Interface Comparison

Codd has classified the languages used to interact with a database management system into three categories, cursor oriented, algebra oriented, and calculus oriented.[30] They

exhibit a hierarchical ordering as depicted below.

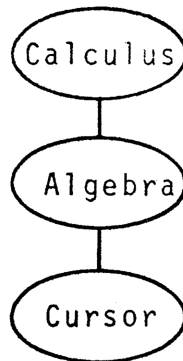



Figure 9

A Cursor oriented language is one in which the user specifies a step by step procedure for extracting the information of interest. In the network model a procedure specification to list the components of an assembly might appear as follows.

```
        Find part xx
        Find first component of part xx
loop:   If not found, go to done
        Print quantity and component number
        Find next component of part xx
        Go to loop
```

A similar procedure would exist for the relational model[31]

```
          Find first tuple of relation component
loop:     If no more tuples, go to done
          If assembly  Part number=xx
          Then Print quantity, Component Part number
          Get next tuple of relation component
          Go to loop.
```

Thus the system is given a systematic procedure via a series of primitive steps for extracting the data of interest.

An algebra oriented language is a concise mathematical abstraction of the procedure to extract information. Functions are provided to operate on aggregate groups of data such as entire relations in the relational model. A single formula (nested perhaps) can express the entire procedure to derive the data of interest. Examples of relational algebra functions can be found in Chapter 3.

A calculus oriented language is one which is free of any procedure specification and merely <u>describes</u> the characteristics of the data desired. The system is required to map the descriptive request into a procedure for data extraction. Examples of calculus oriented requests are to be found in Chapter 6.

Each model of data should have a language at each of the three levels to provide a basis for comparison. Codd has fully developed a relational algebra and a relational calculus.

Network designers have developed a myriad of cursor oriented languages and a handful of network algebras, but a network calculus has yet to be published (see Chapters 5 and 6). Thus, when comparisons are made between the two models in terms of the user interface, they are invariably comparing the relational calculus of Codd with a cursor oriented network language (usually that proposed by the DBTG). The following diagram illustrates the current situation.

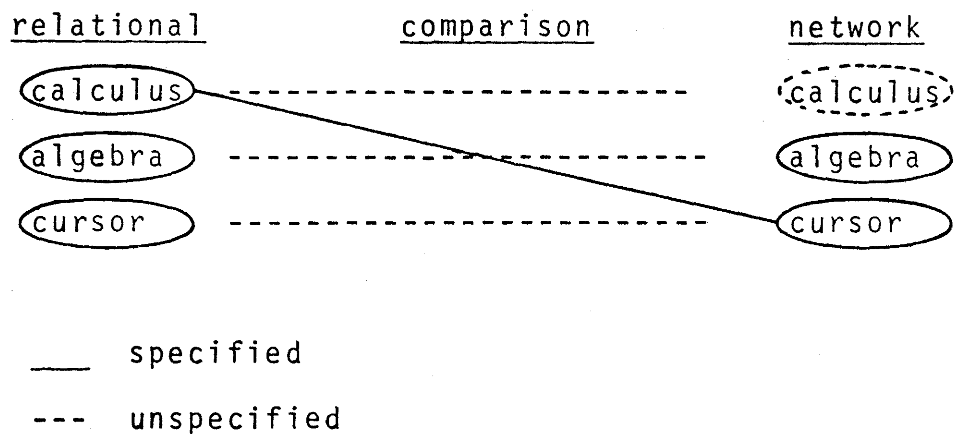



Figure 10

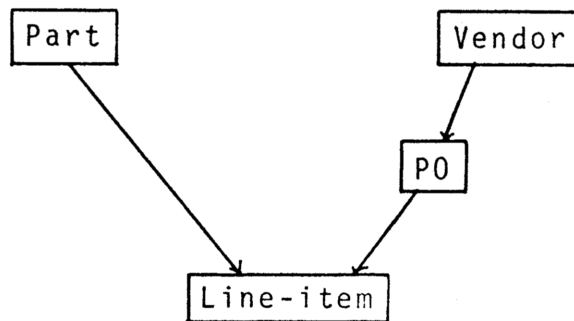
This comparison between the relational calculus and a network cursor oriented language is clearly invalid as a comparison of the two models. To reliably compare the two models of data requires the development of a network algebra and network calculus. Then a more valid comparison between languages on the same level can be made.

Experienced User

An experienced user is one who will interact with a database on a frequent basis, usually as a result of his job. Someone in this category is willing to undertake additional initial education if that will significantly improve the useability of a database management system in the future. A common argument of the relational model advocates is that the relational model is easier to teach to first time users. This is most likely true, for everyone has been presented with a simple table at some time in his life. But this view does not give the user an understanding of the ways individual relations relate to each other; he must be content with the assurance that the system can figure everything out for him. It is the author's contention that a frequent user will desire an understanding of how the database fits together and that the network model will better provide this than the relational model. To achieve the same understanding provided by the network model, the user must be taught the algebraic functions of the relational model. This requires a fairly mathematical mind and makes fully understandingthe relational model a more difficult task.

Due to his frequent encounters with the system, an experienced user is likely to prefer a good algebra oriented language because it provides the most concise expression of his request. His familiarity with the system and under-

standing of the relationships in the database lead him to think of his requests in a procedural fashion. He naturally thinks of the steps required to satisfy his requests, which can in fact be a method whereby be clarifies his request to himself. Because of its more powerful (and hence admittedly more complex) basis, the network model is likely to provide a more powerful algebra oriented language. The user can think in terms of access paths through the database structure, extracting data on the way. A language devised by MITROL, Inc. can provide an example of a good algebra oriented language.[32] Given the following database structure, the command to provide a list of all parts and due-dates on purchase orders under vendor IGG is expressed below. .




ENTER REQUEST: print part-num due-date for vendor IGG po all line-item all.

Figure 11

This request has little superfluous information, and yet it reads quite well. The "for clause" specifies the access path {vendor, po, line-item}, and the "field clause" specifies the requested data. This language, though requiring some understanding of the network model, has proved very successful among MITROL customers.

## Casual User

The bulk of the arguments in favor of the relational model have been oriented to the casual user. Codd has estimated the trend of database usage into the 1990's, and predicts a rapidly growing use by casual users.[33] In his article, "Seven Steps to Rendezvous with the Casual User," Codd defines a casual user as "... one whose interactions with the system are irregular in time and not motivated by his job or social role."[34] Thus he is unwilling to undertake any more than a cursory initial education. If this is true, the simpler the model the better. The argument is then that the easier a model is to teach to a novice, the better it is in fulfilling the needs of a casual user. Now where does the casual user fit into the scheme of the three categories of language interfaces. Clearly he is at the calculus level because he wants to describe his request, not give a procedure for fulfilling it. Codd does an excellent job of describing a system which, based on the relational calculus,

carries on a dialog with the user until the system fully understands the user's request.[35] The user is able to converse with the Rendezvous system in unrestricted English as long as the system can derive some information from his responses. Now find where in the dialog the user needed any concept of a model at all! It seems that the system is the only one using a model, the user merely requests the information he wants. A manager does not need a model of the filing strategies of his secretary, he merely requests the information he wants and the secretary maps the request into the model needed to satisfy it. Thus, indirectly, Codd has led us to the point of saying, from the point of view of the casual user, if a simple model is better than a complex one, then maybe no model is best of all. If this holds true as it would appear to in the sample Rendezvous dialog, then the casual user cannot be used as a basis of comparison between the two models of data. The issue is which model leads to a better interpretation of a user request by the system.

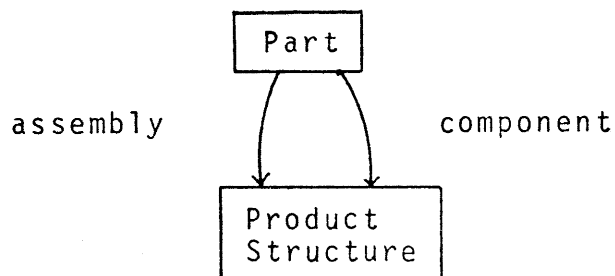
## The System

Without algorithms to decipher a user request in both models, an accurate comparison of the network and relational models in terms of which model leads to easier analysis cannot be made. However, general comments bearing on the issue can be

raised.

Relationships in the network model are information bearing entities absent in the relational model. These relationships have names and it is likely that a user will phrase his English request in terms of these names of relationships. Thus is is likely that these named relationships will make request analysis a simpler job. Note also that a relationship relates a whole entity (all of its attributes) in one file to a whole entity in another file. In the relational model, it could be argued that role names serve the function of relationships, but they do so only indirectly. A role name applies directly only to the foreign key in the relation. It is another step to apply a role name to other domains of tuples found in the other relation.

Consider the part/product structure case. In the network model there are two files with two relations between them.




(only "down" names are given)

Figure 12

In the relational model there would be two relations, part and product structure. The part relation is identical to the part file. The product structure relation has the assembly part number and component part number as well as the quantity field. Consider now the user request, "List all the components of part xx." In the network model this is trivial because the "down" name of one of the relations matches the user's request. In the relational model, the key words given (part and component), do not give the necessary role name for the system to pick up on, assembly. The relational request is really, "List the component part number of all product structures having assembly part number xx. Thus these (network) relation names can provide added ease in deciphering user requests.

Another aspect of the system's job is insuring consistency and integrity of the database. One such problem can be motivated by the following example. Assume there is a vendor relation and a purchase order relation in the relational database. One of the domains of the purchase order relation is likely to be the vendor number. As long as any purchase orders exist for a particular vendor, we would like to disallow the deletion of that vendor from the database. In

the relational model, this enforcement must be external. In the network model it falls out naturally.

Application Programs

One of the major arguments for the relational model is that it provides data independent accessing so that application programs will remain unaffected by growth in the database. In his "Further Normalization of the Data Base Relational Model," Codd gives several examples of cases in which changes to a relational database would impair application programs.[36] The primary cases have to do with attribute migration and insertion and detection anomalies. By using a network model free of ordering and indexing dependencies, the network model can be shown to be no worse than the relational model in its ability to handle growth. (Codd claims the relational model should first be in third normal form, the author claims the network model should first be in a state which accurately reflects the real world.) The bulk of the arguments along these lines deal with particular systems which possess these ordering or indexing dependencies, or treat the network model as a model of storing records, not a model of entities and their relationships.

## Chapter 5 A Network Algebra

### Introduction

There are two possible approaches to be taken in developing a network algebra. The first consists of analyzing usage patterns in a network cursor language and proceeding from there. The second is to take the relational algebra as proposed by Codd, and modify it to fit the network framework. The first approach is preferable if the resulting algebra is an end in itself. If, however, the algebra is to play a support level with respect to the development of a network calculus, then the second scheme is preferable. The issue of the power of the resulting algebra or calculus is important. As will be discussed in the next section, the common approach of following paths through a network database is not powerful enough to handle certain queries. The structure of Codd's relational algebra, as modified here, will take care of these problems. Thus the second approach, modifying Codd's relational algebra to fit the network framework, will be used.

### Access Path Navigation

Bachman has described the programmer's job as one of navigating through the access paths of the network structure.[37] In most network systems, the user begins at the root of the database and, following the arrows of the data structure

diagram, defines a path to the data he requires. In most cases this works quite well, as experienced by MITROL users.[38]

There is one kind of question known to the author in which this navigation procedure fails; there may exist others. Fortunately for network users, this type of question is rare. It will be denoted the "For all" type. Consider the following database structure.

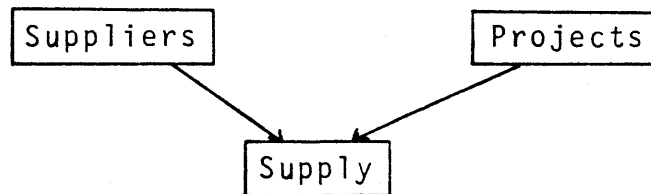



Figure 13

How would one answer the query, "List all the suppliers who supply all projects," by simply navigating via access paths? (This is where division comes in handy.)

## Extension to the Relational Algebra

Setting aside for the moment the one to one and one to many relations of the network model, we see that the remaining files and fields are direct counterparts of the relational relations and domains. Thus we can start with the operations of the relational algebra as a base. To handle the relations we must add a new facility, the MERGE operation.

Let R and S be two network files related by relation T. If r and s are tuples from R and S respectively, then the MERGE of R and S over T is denoted and defined

$$R \overset{T}{\otimes} S \equiv \{r^\frown s : r \varepsilon R \wedge s \varepsilon S \wedge T\ (r,s)\}$$

where T(r,s) is true if r is related to s via relation T.

This can be seen to be the counterpart of the natural join in the relational algebra. The notation given this operation is intended to clarify the fact that the MERGE can be viewed as a limited cartesian product. The operation is clearly the normalization of the two files.

To make the MERGE operation complete, we must extend the notation to include the capability to merge several files at once. A straightforward extension

$$S_1 \overset{T_1}{\otimes} S_2 \overset{T_2}{\otimes} S_3 \overset{T_3}{\otimes} S_4 \ \cdot \ \cdot \ \cdot$$

assumes that the last tuple added is the one taking part in the association of the next merge. If $T_3$, for example, related $S_4$ to $S_2$, then the notation is inadequate. To handle this, the index of the basic tuple in the resulting merge taking part in the merge relation will be placed under the merge symbol. Also, we must have a way to differentiate

between the owner and member side of a one to many relation. To handle this we will place an arrow under the merge symbol. The arrow will point to the right if the file being currently merged is in the member role with respect to the merge relation. For the complementary case it will point left. The full extended notation is thus

$$S_1 \underset{\underset{1}{\rightarrow}}{\overset{T_1}{\otimes}} S_2 \underset{\underset{2}{\leftarrow}}{\overset{T_2}{\otimes}} S_3 \underset{\underset{2}{\rightarrow}}{\overset{T_3}{\otimes}} S_4 \quad . \quad . \quad .$$

For completeness, and to make the reduction from the network calculus as easy as possible, we must also extend the restriction operation of the relational algebra. As proposed by Codd, the restriction operator is defined on the comparison operators $<$, $\leq$, $=$, $\neq$, $\geq$, $>$. We want to add a testing function which determines whether two tuples which have been concatenated (by a cartesian product, for example) are related via a relation. One simple way to do this is to include as part of any tuple a unique identifier, perhaps as the first domain. In the array representation of a relation or file, this might be simply the index. The comparison operator would test these two domains to determine whether their corresponding tuples were related by a given (network) relation. Thus if A and B are the domains identifying tuples r and s in a relation T which has been formed from relations R and S, then

$$T\ [A \overset{W}{\#} B] \quad (T\ [A \overset{W}{\not\#} B])$$

results in a relation of only the tuples of T for which original tuples r and s were (not) related by W. The domain corresponding to the owner tuple will always appear first.

The MERGE operator and the extension to the restriction operator are somewhat redundant. The MERGE operator is preferable in a practical sense because it limits the initial size of a target relation (compared to the full cartesian product). This will become more clear in the network calculus chapter. Due, however, to the lack of mathematical rigor of this presentation, the author is unsure that it will always be sufficient. Thus the restriction modification will be used because it provides mathematical completeness.

Examples

To illustrate the use of the network algebra, consider the following database.

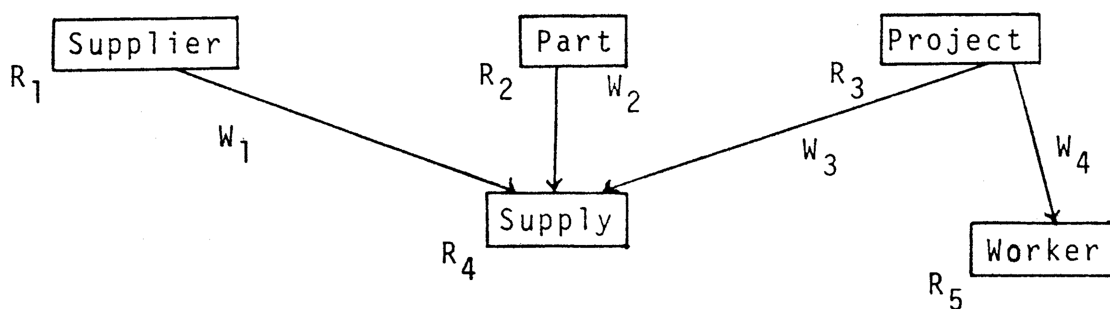



Figure 14

| Symbol | File | Domain 1 | Domain 2 | Domain 3 |
|---|---|---|---|---|
| $R_1$ | supplier | TID* | Supplier # | Supplier Name |
| $R_2$ | part | TID | Part # | Part Name |
| $R_3$ | project | TID | Project # | Location |
| $R_4$ | supply | TID | | |
| $R_5$ | worker | TID | Worker # | Worker Name |

* Tuple Identifier

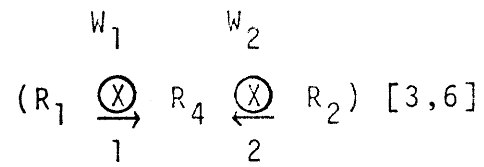
Figure 14 (cont'd)

The following queries will be translated into the network algebraic formulas required to satisfy them.

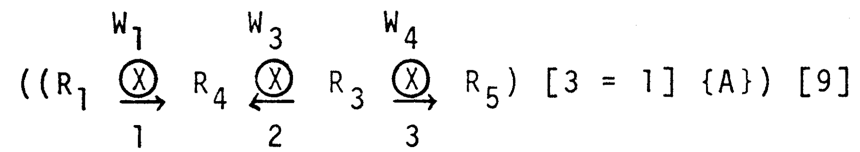
* Find the supplier numbers of those suppliers who supply part 15.

$$((R_1 \underset{1}{\overset{W_1}{\underrightarrow{\otimes}}} R_4 \underset{2}{\overset{W_2}{\underleftarrow{\otimes}}} R_2)\ [6 = 1]\ \{15\})\ [2]$$

* Find the name of suppliers and the parts being supplied by them (omitting those suppliers who are supplying no parts at this time).



*     Find the workers on all projects supplied by supplier A.



## Chapter 6 A Network Calculus

### Introduction

Just as with the network algebra, the network calculus presented here will be a simple extension of the relational calculus developed by Codd.[39] The presentation will closely follow his, but with the slight modification included. Because the presentation given here is fairly terse, the reader is suggested to refer to Codd's presentation for clarification.[40]

### The Extension

All that is necessary to convert Codd's relational calculus to a network calculus is the addition of dyadic predicate constants $W_1$, $W_2$, $W_3$, ... to the alphabet, one for each (network) relation in the database. Throughout the development, they can be handled exactly as the dyadic predicate constants $<$, $\leq$, $=$, $\neq$, $>$, $\geq$ are, except that they take tuple variables as opposed to indexed tuples on either side. We shall follow the convention that the owner tuple shall appear on the left.

### The Network Calculus

The alphabet for the network calculus is listed below.

| | |
|---|---|
| Individual constants | $a_1$, $a_2$, $a_3$, ... |
| Index constants | 1, 2, 3, 4, ... |
| Tuple variables | $r_1$, $r_2$, $r_3$, ... |
| Predicate constants | |
| monadic | $P_1$, $P_2$, $P_3$, ... |
| dyadic | =, <, >, $\leq$, $\geq$, $\neq$ |
| | $W_1$, $W_2$, $W_3$, ... |
| Logical symbols | $\exists$, $\forall$, $\wedge$, $\vee$, $\neg$ |
| Delimiters | [] (), |

There is a monadic predicate constant for each file (relation) in the database, and $P_j r$ is intended to mean that tuple r is a member (row) of file j. $P_j r$ is called a range term.

An indexed tuple, denoted r[n] where r is a tuple variable and n is an index constant, is used to identify the $n^{th}$ domain of tuple r.

If $\theta$ is one of the dyadic predicate constants =, <, $\leq$, >, $\geq$, $\neq$, and $\lambda$ and $\mu$ are indexed tuples, then $\lambda\theta\mu$ and $\lambda\theta\alpha$ are called join terms. If $\theta$ is one of the dyadic predicate constants $W_1$, $W_2$, $W_3$, ..., and $\lambda$ and $\mu$ are tuples, then $\lambda\theta\mu$ is called a merge term. The only terms of the network

calculus are range terms, join terms, and merge terms.

Codd's definition of a well-formed formulae (WFF) still applies

"1. Any term is a WFF;

2. If $\Gamma$ is a WFF, then so is $\neg\Gamma$;

3. If $\Gamma_1, \Gamma_2$ are WFFs, so are $(\Gamma_1 \wedge \Gamma_2)$ and $(\Gamma_1 \vee \Gamma_2)$;

4. If $\Gamma$ is a WFF in which r occurs as a free variable, then $\exists r(\Gamma)$ and $\forall r(\Gamma)$ are WFFs;

5. No other formulae are WFFs."[41]

A range WFF is a quantifier free WFF whose only terms are range terms. A range WFF over r is a range WFF with r as the only free variable. A proper range WFF over r is a range WFF over r such that:

1. $\neg$ occurs only after $\wedge$, and
2. if r is part of more than one range term, the relations associated with the predicates are union compatible.

Thus a proper range WFF over r cannot say only that file S is not the source of tuple variable r. Likewise the range of a tuple variable can only be the files of the database or files which can be generated from them by union, intersection,

and difference operations on union compatible pairs.

A range-coupled quantifier is either $\exists\Gamma r$ or $\forall\Gamma r$ where $\Gamma$ is a proper range WFF over r.

A range-separable WFF can be written in conjunctive form

$$U_1 \wedge U_2 \wedge \ldots \wedge U_n \wedge V,$$

where

"1. $n \geq 1$;

2. $U_1$ through $U_n$ are proper range WFFs over n distinct tuple variables;

3. V is either null, or it is a WFF with the three properties:

   a. every quantifier in V is range-coupled;

   b. every free variable in V belongs to the set whose ranges are specified by $U_1, U_2, \ldots, U_n$;

   c. V is devoid of range terms."[42]

To incorporate the needed projection capability, a simple alpha expression has the form

$$(t_1, t_2, \ldots, t_k) : w$$

where

1. $w$ is a range-separable WFF of the network calculus;
2. $t_1, t_2, \ldots, t_k$ are distinct terms, each consisting of a tuple variable or an indexed tuple variable;
3. the set of tuple variables occurring in $t_1, t_2, \ldots, t_k$ is precisely the set of free variables in $w$.

An alpha expression is either a simple alpha expression, or, if $t{:}w_1$ and $t{:}w_2$ are alpha expressions, an expression of the form

$$t{:}(w_1 \vee w_2)$$
$$t{:}(w_1 \wedge w_2)$$
$$t{:}(w_1 \wedge \neg w_2).$$

Examples

Assume the following database structure.

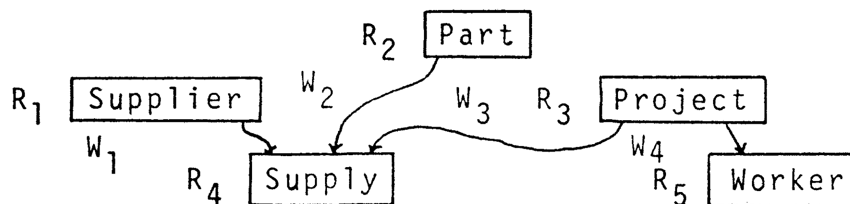



Figure 15

| Symbol | File | Attribute 1 | Attribute 2 | Attribute 3 |
|---|---|---|---|---|
| $R_1$ | supplier | supplier # | supplier name | location |
| $R_2$ | part | part # | part name | |
| $R_3$ | project | project # | | |
| $R_4$ | supply | | | |
| $R_5$ | worker | worker # | worker name | |

Figure 15 (cont'd)

* Find the supplier number of those suppliers who supply part 15.

  $r_1[1]:P_1 r_1 \wedge P_2 r_2 \wedge P_4 r_4 \wedge (r_1 W_1 r_4 \wedge$
  $r_2 W_2 r_4 \wedge r_2[1] = 15)$

* Find the locations of suppliers and the parts being supplied by them (omitting those suppliers who are supplying no parts at this time).

  $(r_1[3], r_2[1]) : P_1 r_1 \wedge P_2 r_2 \wedge P_4 r_4 \wedge$
  $(r_1 W_1 r_4 \wedge r_2 W_2 r_4)$

* Find the workers on all projects being supplied by supplier A.

$$r_5[1] : P_1 r_1 \wedge P_4 r_4 \wedge P_3 r_3 \wedge P_5 r_5 \wedge (r_1[1] = A \wedge r_1 W_1 r_4 \wedge r_3 W_3 r_4 \wedge r_3 W_4 r_5)$$

Reduction

The reduction algorithm presented by Codd is intended to demonstrate that the relational algebra is "relationally complete" and is not intended as a practical efficient translator from calculus to relational algebra.[43] Because the modification we have presented is of such a minor nature, we will not wade through a parallel reduction algorithm for the network calculus and algebra. Instead, we will present arguments considered sufficient to convince the reader that such a parallel reduction exists, and move on to the more interesting question of efficiency.

The only modification we made to the relational calculus was the addition of the dyadic predicate constants $W_1$, $W_2$, $W_3$, ... These operated on tuples in an identical manner to the way the other dyadic predicate constants $(=, \neq, <, \leq, >, \geq)$ operated on indexed tuples. By assuming that we will assign each tuple in a file a unique identifier (as discussed in the last chapter) and treating that as the first domain of each tuple, then the extension we made to the restriction

operation in the last chapter will handle the extension we made to the network calculus.

To be specific, if we assume that the tuple identifier domain is not assumed in the calculus level, but that the cartesian product automatically prefixes it to the tuple, then the following modifications need to be added to the reduction algorithm.

Step 1.3

When a dyadic predicate constant W is preceded by a $\neg$, eliminate the $\neg$ symbol and replace W by $\overline{W}$.

Step 3

$$\mu_j = (\sum_{i=1}^{j-1} (n_i + 1)) + 1 \qquad \text{(change)}$$

Step 4 (rewriting rules)

5. $(r_j \; W \; r_k) \rightarrow S[\mu_j - 1 \overset{W}{\#} \mu_k - 1]$ (add)

6. $(r_j \; \overline{W} \; r_k) \rightarrow S[\mu_j - 1 \overset{W}{\not\#} \mu_k - 1]$ (add)

## Optimization and Efficiency

It would be unfair at this point in the development to declare that one model or the other is inherently more efficient. Codd argues that the calculus level is a good starting point for optimization, and this is likely to be true. The issue of concern here is whether the network model or relational model leads to more efficient execution.

The use of the cartesian product is a prime suspect with which to begin. The number of tuples generated by an extended cartesian product can quickly become astronomical. The cartesian product of three relations with only 100 tuples each will result in a relation with 1,000,000 tuples. A majority of these are likely to be pared off in the ensuing restriction operations.

It is likely that with a little effort, the cartesian products and restrictions could be replaced by joins. Note that

$$R[A \theta B]S = (R \otimes S) [A \theta B].$$[44]

So the more likely efficiency comparison should be between the join of the relational algebra and the merge of network algebra. In either case improvement could also result from performing some restrictions prior to the merge or join. In a likely implementation of the network model, the merge operation should require no more accesses than the number of tuples which would end up in the resulting file. The author does not know of a relational implementation which would have this same property, but this is more likely a function of the ignorance of the author than the non-existence of such an implementation. Thus we cannot pursue this point any further.

## Chapter 7 Summary

### Conclusions

Once again, the subjective nature of a comparison of this type must be stressed. Any conclusions drawn must be phrased in terms of the issues discussed. The relevancy and importance of the topics chosen for study here are matters for the reader to evaluate. The author made an attempt to cover the areas of concern most frequently found in the literature, and this may be some basis of argument for the relevancy of the topics. It is the author's personal opinion that the discussion has hit many of the important issues and has done so with a viewpoint not commonly found in current debates.

Based on the five viewpoints chosen for comparision, all but the casual user point of view leaned toward a preference for the network model; for the casual user it was a draw. This is perhaps an appropriate point to emphasize that all aspects of the network model used for the comparisons are not, to the author's knowledge, to be found in any published exposition of the model. Some may argue that this makes the comparisons invalid, but the author contends rather that looking at each model in terms of its ultimate possibilities makes the comparison more meaningful, if the goal is in fact to choose the best approach. Rather than focussing on correctible deficiencies in a current model, we have tried to look at the underlying

and inherent characteristics of each model with the hope that the results will not become outdated as soon as a particular deficiency is remedied.

To provide an adequate basis of comparison with respect to the interface language, we have followed Codd's lead and developed a network algebra and network calculus (which were simply extensions to Codd's). The approach was to a certain extent non-rigorous, and it is suspected that this may provide, for some, a point of contention. The hope is that the extensions proposed will serve as the basis of a more mathematical treatment by anyone with an abstract mathematical background. More importantly, we hope that this will provide an impetus toward more "equal basis" comparisons in the future.

One question that arises is whether one model can be viewed as a subset of another. An excerpt from Codd has bearing on this issue.

> "Claims have been made ... that the network approach ... permit(s) more natural or faithful modelling of the real world than the relational model. Such claims are not easy to support or refute, because our present knowledge of what constitutes a good data structure for solving a given class of problems is highly intuitive and unsystematic.
>
> "However, we can observe that many different kinds of

geometry, topology, and graphs (or networks) are in use today for solving "real world" problems. Relations tend to be neutral towards these problem-solving representations and yet very adaptable to supporting any of them. This has been demonstrated rather clearly in applications of relations in various kinds of graphics packages.

"On the other hand, the owner-coupled set gives rise to a specific kind of network, and is accordingly very convenient in some contexts and very awkward in others. It is convenient when the application involves collections of sets, each of which has both a descriptor and a simple total ordering of its elements. It is awkward when the application involves partial orderings (e.g., PERT charts), loops (e.g., transportation routes), values associated with network links (e.g., utility networks), many to many binary relations, relations of degree other than two, and variable depth, homogeneous trees (e.g., organization charts)."[45]

Codd proceeds to argue that the principal schema should be kept simple, and particular user needs for more complex schemes should be the responsibility of the user schema. This, he argues, would provide a clean separation that would keep the principal schema uncluttered. He is thus saying that the network model is a structure which can be built on top of the relational model. This could well be true, but the relational model is trying to

accomplish the same ends as the network model, and hence skirts around the network model. The thesis underlying this paper has been that the network model, as presented here, is sufficient to model the real world. As Codd points out, this is difficult to support or refute, but, if it is true, the network model provides a higher interface to use and hence should provide greater ease in handling real world problems.

A Hybrid View

Codd points out that the network view can be supported by the relational model. It is likewise possible to support the relational view in a system built on the network model. The basis of this assertion comes from the fact that there exists a procedure (a la Codd) for transforming a network database into a relational database (normalization). Without actually performing this normalization, the network system could make it appear to the user that this has been done and that the database is built on the relational view. Thus, if it behooves one to think of a database as relational (e.g., the casual user), there is no reason to rule out the support of it even if the system has been built on a network framework. With this possibility we can have the best of both worlds.

A special case in which the hybrid view seems appropriate is in the context of distributed databases. Consider a user database which maintains information on his current portfolio.

An attribute of major concern with respect to a stock is its current price. This is an extremely expensive value to keep track of continuously, especially for a single user. It would more likely be desirable to have a single vendor (user) maintain current prices on all stocks, and any individual user could, at any time, access the current price of a particular stock (for a slight fee, of course). It may be helpful to view an individual user's database in the network model, and the interaction between databases in a relational context. To find the stocks in the user's portfolio we would employ network techniques, and to find the stock price in a remote database we would make use of relational operators.

## Further Research

This paper has touched on several ideas which would themselves make interesting bases for research. The scope of this paper was much too broad to cover many of them in any detail. A few of these are suggested below.

* The whole concept of keys in the network model is a source of problems in several implementations. As mentioned in Chapter 2, keys serve two functions in the network model, ordering and identification of members of a relation, and identification of records in a file. In the hierarchical model of data, out of which the network

model evolved, this duality of function caused no problems. It is the potential for multiple access paths to a record that spawns the problem, and the duality of function should thus be clearly separated. How this can be done, and its impact on current use of the network model would be an interesting study.

* As mentioned in Chapter 2, one to many binary relations can be used to represent many to many and n-ary relations. A mathematical formulation and proof of this assertion could prove interesting. Also the need for one to one relations alluded to could be verified.

* A psychological study of the "teachability" of the two models could help to reinforce or quell many arguments.

* Many of the ideas presented in developing the network algebra and calculus could stand a more formal, mathematical treatment.

* Techniques for optimization of both the network and relational calculus in the reduction process would be of significant practical value.

* Any comparisons of the two models on terms other than those presented here would help to round out the picture and fill in the gaps. These should not have any of the pitfalls discussed in the introduction.

## Footnotes

1. Codd, E.F., "Seven Steps to Rendezvous with the Casual User," Proceedings of the IFIP TC-2 Working Conference on Data Base Management Systems, Cargese, Corsica, April 1-5, 1974, North-Holland, Amsterdam.
2. Clemons, Eric K., "The Design of Languages for Management Information Systems: A Proposal for a Disseration," Cornell University, March 12, 1975.
3. Madnick, Stuart E., "The Future of Computers," Technology Review, July/August, 1973.
4. Simon, H.A., "The Architecture of Complexity," Proceedings of the American Philosophical Society, 106, 6, December, 1962, pp 467-482.
5. Codd, E.F., "A Relational Model of Data for Large Shared Data Banks," Communications of the ACM, Volume 13, Number 6, June, 1970, pp 377-387.
6. Bachman, C.W., "Data Structure Diagrams," Data Base (Quarterly News Letter of ACM-SIGBDP) Volume 1, Number 2, 1969.
7. Bachman, C.W., "The Data Base Set Concept: Its Usage and Realization," Honeywell Information Systems Internal Report, January 31, 1973.
8. Bachman, C.W., "The Programmer as Navigator," Communications of the ACM 16, No. 11, November 1973, pp 653-658.
9. Op. Cit., Codd, E.F., "A Relational Model of Data for Large Shared Data Bases."

10. Codd, E.F., "Further Normalization of the Data Base Relational Model," Courant Computer Science Symposia 6, "Data Base Systems," New York City, May 24-25, 1971, Prentice Hall.

11. Codd, E.F., "Normalized Data Base Structure: A Brief Tutorial," Proceedings of the 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control, San Diego, available from ACM, New York.

12. Codd, E.F., "Relational Completeness of Data Base Sublanguages," Courant Computer Science Symposia 6, "Data Base Systems," New York City, May 24-25, 1971, Prentice Hall.

13. Codd, E.F., and C.J. Date, "Interactive Support for Non-Programmers: The Relational and Network Approaches," Proceedings of the 1974 ACM-SIGFIDET Workshop, available from ACM, New York.

14. Op. Cit., Codd, E.F., "A Relational Model of Data for Large Shared Data Banks."

15. Ibid

16. Ibid

17. Op. Cit., Codd, E.F., "Relational Completeness of Data Base Sublanguages."

18. Ibid

19. Ibid

20. Op. Cit., Codd, E.F., "Further Normalization of the Data Base Relational Model."

21. Op. Cit., Codd, E.F., "Normalized Data Base Structure: A Brief Tutorial."

22. Op. Cit., Codd, E.F., "Relational Completeness of Data Bases Sublanguages."

23. Ibid

24. Ibid

25. Ibid

26. Op. Cit., Codd, E.F., "Seven Steps to Rendezvous with the Casual User."

27. Op. Cit., Codd, E.F., "Further Normalization of the Data Base Relational Model."

28. Ibid

29. Smith, Grant N., "Decision Rules for the Automated Generation of Storage Strategies in Data Management Systems," Sloan School of Management, Masters Thesis, Cambridge, Massachusetts, June 1975.

30. Op. Cit., Codd, E.F., "Relational Completeness of Data Base Sublanguages."

31. Op. Cit., Codd, E.F., and C.J. Date, "Interactive Support for Non-Programmers: The Relation and Network Approaches."

32. MITROL, Inc., "MITROL Technical Reference, The Print Request," available from MITROL, Inc., Waltham, Ma.

33. Op. Cit., Codd, E.F., "Seven Steps to Rendezvous with the Casual User."

34. Ibid

35. Ibid

36. Op. Cit., Codd, E.F., "Further Normalization of the Data Base Relational Model."

37. Op. Cit., Bachman, C.W., "The Programmer as Navigator."

38. Op. Cit., MITROL, Inc., "MITROL Technical Reference, The Print Request."

39. Op. Cit., Codd, E.F., "Relational Completeness of Data Base Sublanguages."

40. Ibid

41. Ibid

42. Ibid

43. Ibid

44. Ibid

45. Op. Cit., Codd, E.F., and C.J. Date, "Interactive Support for Non-Programmers: The Relational and Network Approaches."

Bibliography

1. Clemons, Eric, K., "The Design of Languges for Management Information Systems: A Proposal for a Dissertation," Cornell University, March 12, 1975.

2. Madnick, Stuart E., "The Future of Computers," Technology Review, July/August, 1973.

3. Simon, H.A., "The Architecture of Complexity," Proceedings of the American Philosophical Society, 106, 6, December, 1969, pp 467-482.

4. Bachman, C.W., "The Data Base Set Concept: Its Usage and Realization," Honeywell Information Systems Internal Report, January 31, 1973.

5. Bachman, C.W., "Data Structure Diagrams," Data Base (Quarterly News Letter of the ACM-SIGBDP) Volume 1, Number 2, 1969.

6. Bachman, C.W., "The Programmer as Navigator," Comm. ACM 16, No. 11, November 1973, pp 653-658.

7. Codasyl Development Committee, "An Information Algebra," CACM 5, April 1962, pp 190-204.

8. CODASYL, "Data Base Task Group Report," ACM HQ, October 1969.

9. Codasyl Systems Committee, "Feature Analysis of Generalized Data Base Management Systems," May 1971, available from ACM, New York.

10. CODASYL, "Data Base Task Group Report," ACM, New York, April 1971.

11. CODASYL Data Base Language Task Group: Proposal, February 1973.

12. Codd, E.F., "A Relational Model of Data for Large Shared Data Banks," Communications of the ACM, Volume 13, Number 6, June 1970.

13. Codd, E.F., "Further Normalization of the Data Base Relational Model," Courant Computer Science Symposia 6, "Data Base Systems", New York City, May 24-25, 1971, Prentice-Hall.

14. Codd, E.F., "A Data Base Sublanguage Founded on the Relational Calculus," Proceedings of the 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control, San Diego, available from ACM, New York.

15. Codd, E.F., "Normalized Data Base Structure: A Brief Tutorial," Proceedings of the 1971 ACM-SIGFIDET Workshop on Data Description, Access, and Control, San Diego, available from ACM, New York.

16. Codd, E.F., "Relational Completeness of Data Base Sublanguages," Courant Computer Science Symposia 6, "Data Base Systems," New York City, May 24-25, 1971, Prentice-Hall.

17. Codd, E.F., "Seven Steps to Rendezvous with the Casual User," Proceedings of the IFIP TC-2 Working Conference

on Data Base Management Systems, Cargese, Corsica, April 1-5, 1974, North-Holland, Amsterdam.

18. Codd, E.F., "Recent Investigations in Relational Data Base Systems," Information Processing 74, North-Holland, Amsterdam.

19. Codd, E.F., and C.J. Date, "Interactive Support for Non-Programmers: The Relational and Network Approaches," Proceedings of the 1974 ACM-SIGFIDET Workshop, available from ACM, New York.

20. Date, C.J., and E.F. Codd, "The Relational and Network Approaches: Comparison of the Application Programming Interfaces," Proceedings of the 1974 ACM-SIGFIDET Workshop, available from ACM, New York.

21. Smith, Grant N., "Decision Rules for the Automated Generation of Storage Strategies in Data Management Systems," Sloan School of Management, Masters Thesis, Cambridge, Massachusetts, June, 1975.

22. MITROL, Inc., "MITROL Technical Reference, The Print Request," available from MITROL, Inc., Waltham, Ma.

23. Bachman, C.W., "The Programmer as Navigator," Comm. ACM 16, No. 11, November 1973, pp 653-658.

24. Stamen, J.P., and R.M. Wallace, "Janus - A Data Management and Analysis System for the Behavioral Sciences," Cambridge Project, Cambridge, Ma.