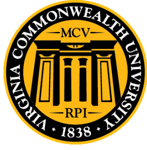


Virginia Commonwealth University
VCU Scholars Compass

Theses and Dissertations | Graduate School

1985

Designing a Relational Data Base for a Problem Solving Environment

Kathryn S. Dawson

Follow this and additional works at: https://scholarscompass.vcu.edu/etd

Part of the Computer Sciences Commons

© The Author

Downloaded from
https://scholarscompass.vcu.edu/etd/4508

This Thesis is brought to you for free and open access by the Graduate School at VCU Scholars Compass. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of VCU Scholars Compass. For more information, please contact libcompass@vcu.edu.

المنارة للاستشارات

www.manaraa.com

College of Humanities and Sciences
Virginia Commonwealth University

This is to certify that the thesis prepared by Kathryn S. Dawson entitled Designing a Relational Data Base for a Problem Solving Environment has been approved by her commmittee as satisfactory completion of the thesis requirement for the degree of Master of Science in Mathematical Sciences/ Computer Science.

Dr. Lorraine M. Parker
Director of Thesis

Dr. Nancy Jacqmin
Committee Member

Dr. Francis Kane
Committee Member

Dr. James Wood
Director of Graduate Study

Dr. William E. Haver
Department Chairman

Dr. Elske v. P. Smith
Dean, College of Humanities and Science

MAY 14, 1985
Date

DESIGNING A RELATIONAL DATA BASE FOR A PROBLEM SOLVING ENVIRONMENT

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at Virginia Commonwealth University

By

Kathryn S. Dawson

Director: Dr. Lorraine M. Parker
Assistant Professor of Mathematical Sciences
Virginia Commonwealth University

Virginia Commonwealth University
Richmond, Virginia

May, 1985

ACKNOWLEDGEMENT

The author would like to thank her advisor, Dr. Lorraine Parker for all her excellent advice and continuing encouragement throughout the preparation of this document.

The author would also like to thank Dr. Francis Kane and Dr. Nancy Jacqmin for their valuable editorial comments and support.

Table of Contents

Page

List of Tables v

List of Figures vi

Abstract ix

Chapter 1 Introduction 1

Chapter 2 Definition of a Data Base 3

Interrelated Data 5
Redundancy Eliminated 5
Data Independence 6
Common Controlled Approach 8
Components of a Data Base System 9
Advantages 16

Chapter 3 Logical View of a Data Base 22

Hierarchical 23
Network 28
Relational 32
Advantages of the Relational Model 36

Chapter 4 Physical Design of a Data Base 45

Storage Space Available 46
Volatility of a File 47
Speed of Retrieval 49
Multiple Indexes 51
Logical Design 52

Chapter 5 The Relational Data Base Model 55

Relational Constructs 55
Relational Algebra 66
Null and Default Values 98

Chapter 6 Normal Forms 107

First Normal Form 108
Functional Dependencies 109

Second Normal Form 112
Third Normal Form 114
Nonloss Decomposition of Relations 116
Boyce/Codd Normal Form 120
Fourth Normal Form 123
Fifth Normal Form 127

Chapter 7 Development of a Data Base 133

Description of the Problem 133
A Relational Data Base as a Basis for
Scheduling System Design 137
Data Needed to Solve the Problem 139
Constructing the Logical View 145
Verifying the Logical Design 156
The Physical Design 165
Design Implementation 174

Chapter 8 Conclusion 183

Bibliography 185

Appendix A Data Dictionary 186

Appendix B Implementation Algorithms 194

Vita 201

List of Tables

Table Page

1. Physical Design Tradeoffs 53

2. Comparison of Execution Times in Seconds on Various Data Structures 169

List of Figures

Figure Page

1. Components of a Data Base System 10
2. Logical Record Definition 15
3. Hierarchical View of Library Data 24
4. Network View of Library Data 29
5. Relational View of Library Data 33
6. CURRENT_EMPLOYEES Relation 56
7. LOCATION Relation 60
8. PROMOTIONS Relation 62
9. Family Information 63
10. FAMILY Relation 64
11. EDUCATION Relation 66
12. PROG Relation 67
13. HISTORY Relation 69
14. NUMS Relation 70
15. LOC Relation 71
16. BOTH Relation 73
17. YOUNG_SON Relation 75
18. APPLICANT and AVAILABLE Relations 77
19. ALL Relation 78
20. SUPERVISORS Relation 80
21. INFO Relation 81

22. ASSIGNMENTS Relations 82

23. MORE Relation 83

24. RESEARCH and EM_RELATIONS Relations 84

25. BOTH_COMM Relation 84

26. CONTRACTS Relation 94

27. CONTRACT2 Relation 94

28. OFFICE Relation 95

29. OFFICE2 Relation 96

30. OP and SURG Relation 99

31. MAYBE_OP Relation 100

32. PAT_SURG Relation 101

33. PAT_SURG2 Relation 102

34. SKILLS Relation 108

35. CO_INFO Functional Dependency Diagram 111

36. Decomposition of CO_INFO Relation 114

37. Decomposition of JOB_INFO Relation 116

38. Alternative Decomposition of JOB_INFO 117

39. Alternative Decomposition of JOB_INFO 118

40. Functional Dependency Diagram for AGNTS Relation 121

41. Decomposition of AGNT Relation 122

42. Functional Dependency Diagram for AGNT2 Relation 123

43. Table of Computer Skills 124

44. SKILLS Relation 124

45. AGNT Relation 127

46. Invalid Decomposition of Relation AGNT 128

47. Valid Decomposition of AGNTS 129

48. Relationships Among Faculty Data Items 141

49. Relationships Among Classes Data Items 144

50. Relationships Among All Data Items 146

51. Modification of SS# and Time Not Available Relationship 148

52. Functional Dependencies in FACULTY Relation 156

53. NOT_AVAIL Functional Dependencies 157

54. SCHEDULE Functional Dependencies 157

55. CLASSES Functional Dependencies 158

56. PERSONAL Functional Dependencies 159

57. Multivalued Dependencies 161

58. Key Attribute Relationships in SCHEDULE 162

59. Possible Decomposition of PERSONAL 163

60. Contents of Data Base Account 175

61. Login Menu 176

62. Sample PL/I Program 178

Abstract

When chosing a system design in which to solve a recurring problem which depends on interrelated data a relational data base environment should be considered. The original problem can be solved through this design and by allowing users to view the data in the relational constructs the data can be easily used in numerous other applications. Theorical support insures the design is sound avoiding inaccurate results. Independence between the logical and physical views of the data enables the data base administrator to adjust the physical data structures in order to optimize system performance without affecting existing user applications.

CHAPTER 1

INTRODUCTION

When developing a computer system to solve a particular problem one consideration is the environment in which the problem will be solved. If the problem addresses an issue that will not be reexamined at a later time the system can be designed so it will efficiently present the solution to the problem in the quickest amount of time. If, however, the problem is such that its solution will be required under a variety of circumstances the system's design should include a mechanism whereby data values can be easily updated. The problem can then be resolved using these new values.

Another consideration could be whether the data involved in the solution of the problem can be used to solve other problems. If this is the case the system should be designed so that the definition of the data available is understood by all potential users. Also if additional data items are needed by these new problems the system ideally should be designed so that this

expansion of the defintion of the data sets will not corrupt existing applications.

A relational data base system provides an environment through which a particular problem can be solved as well as numerous others that rely on the same data. If a partiular problem requires additional information the collection of data can be easily expanded to provide for these new values. While the initial design of this type of system could take longer to implement than a system which addresses only the initial problem, the long term benefits it could provide offset this consideration.

CHAPTER 2

DEFINITION OF A DATA BASE

A data base is a way of organizing and maintaining information in a computer. Before precisely defining a data base it is advantageous to consider a situation where the principles of data base theory are not employed.

Suppose a small company has a computer available for its various departments to use. The warehouse department has developed several programs and files which supply information about parts on hand and ordered. The manufacturing department has developed their own programs and files which provide information about parts needed in the manufacturing of each product line. Of course each department designed their systems in their own independent ways. The warehouse department files might be indexed by part number and contain no information about the product which calls for each part. The manufacturing department files could be indexed by

product number and contain only a character description of parts needed. Each department's system is satisfactory for its particular needs. Now suppose that management wants to do a productivity study that involves data from both departments. Since the files used by the two departments are probably incompatible, a programmer would have to start from the basic level of building files which will provide the same data as is now available. At best this will be a time consuming task which, if the data had been jointly maintained, could have been avoided. A more satisfactory approach would have been to have a centralized system of organized data files. Each department could be responsible for updating the data that pertains to its department and still have the means for maintaining the types of reports pertinent to its operation. But now an overall view of all the data would be available for future unanticipated uses. A system designed to solve these problems is a data base.

A data base is a collection of interrelated data stored such that 1) harmful or unnecessary redundancy is eliminated, 2) applications and user access methods are independent of the data structures used, and 3) a common, controlled approach is used to update, delete and retrieve data[1].

INTERRELATED DATA

When a user is utilizing a large collection of data it is important to be able to retrieve not only the data itself but the relationships that exist among data items[2]. Suppose, for instance, in the small company example that two files were available, one listing the numbers of all parts currently stocked and another listing all products currently being manufactured. What is missing is a link between these two files which would inform a user which parts are needed for each product. Clearly the relationship between the data items stored is as important as the data itself and should be represented as well. This can be accomplished in several ways as will be in seen in Chapter 3.

REDUNDANCY ELIMINATED

Consider again the original small company example but suppose now that the part number rather than the part description is recorded in the manufacturing department's file. In this case the part numbers are listed in the manufacturing's and warehouse's separately maintained systems. A high degree of cooperation is needed, however, between the two departments to insure that the part numbers are correct in both systems. If

the warehouse department decides for instance, to order a part from another company resulting in a new part number the manufacturing department must be informed of this change. If the management of the company asks for an inventory by part number separate listings could be made of parts in the warehouse and in the manufacturing departments. The accuracy of the total inventory, however, depends on having the part numbers correct in both files. If any changes made in part numbers are not accurately and quickly recorded in both departments the integrity of the inventory report will be challenged.

Clearly the fact that part numbers are stored in two separately maintained systems jeopardizes the correctness of any report that relies on their joint accuracy[3]. In a data base system the elimination of redundant data is strived for. In cases when this is not advantageous a mechanism is provided so that the redundant data is updated simultaneously[4]. This requires a knowledge of what the redundant data is and a software mechanism for insuring that a requested update to this data is made wherever it is listed.

DATA INDEPENDENCE

In order to describe data independence it is

helpful to consider data dependence. In the manufacturing company example each department separately designed their file structures. The warehouse department's programmer might have preferred files indexed by part number. All the programs that access that file are dependent on that structure. If the programmer decided later to redesign the data structure all of the programs that access that file would have to be updated. This would include programs that simply generate reports as well as those that are used to update the file.

Clearly, when a new data structure is called for, a large amount of time will be needed to modify and test each application program. Data independence means that the physical design of the data base can be changed without modifying the user's view of the data or their application programs[5]. To meet this criteria a data base system provides two levels of software. The level the end user employs is independent of the actual data structure used. This can be accomplished through a second level of software or mapping routines which interface with the actual physical files though a series of subroutines. The end user should be oblivious to this level. The numerous application programs would call these subroutines in a consistent way. When the data base designer decides to restructure the data only these subroutines would have to be modified. The end

user's method of accessing the data base would remain the same, avoiding confusion and time consuming program updates.

COMMON CONTROLLED APPROACH

In the manufacturing company example each department's files were designed separately and independently. Each system would provide methods of updating and accessing these files but in what could be very different ways. If an employee of the warehouse department were to access the manufacturing department's data a new approach and understanding of the data structures would be required. For an inexperienced user this could prove cumbersome. The result is that valuable information available in one department would be difficult for personnel in other departments to access.

A data base system provides a common approach for all users to employ in order to access and modify data[6]. In this way information pertinent to one department can be available without difficulty to any authorized user. Of course some data must be protected. For instance if the personnel department had a file containing employee salaries they would not want this information to be accessible to all users. Similarly, only authorized

employees should be allowed to modify the data. The data base designer should provide a mechanism to insure that access is controlled.

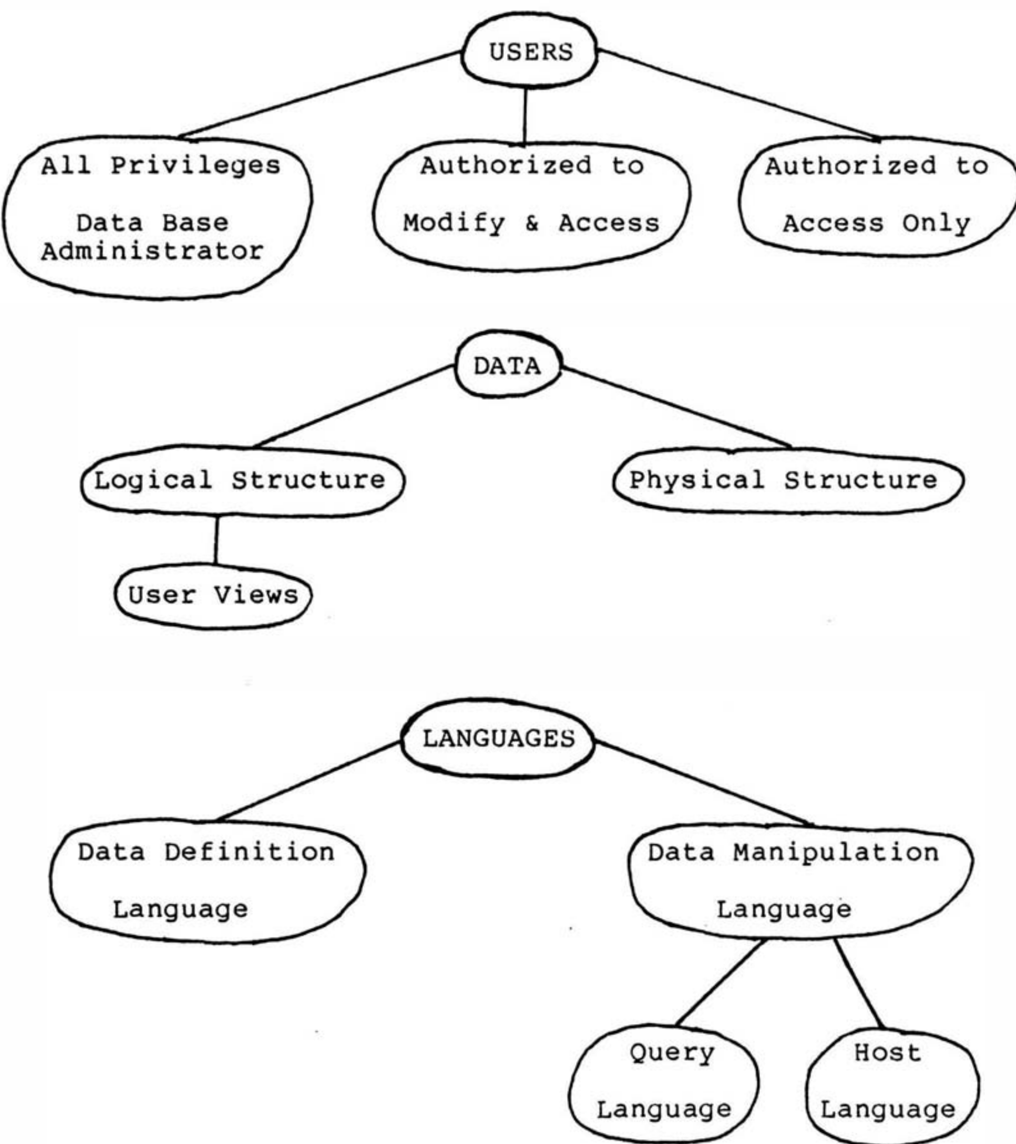
COMPONENTS OF A DATA BASE SYSTEM

Throughout the previous discussion several aspects and levels of a data base have been mentioned. Figure 1 illustrates how these various pieces fit together to form a complete data base system.

Users

One way the users of a data base system can be classified is based on the privileges they have. The data base administrator (DBA) has all privileges and can be considered the guardian of the system. It is this person who sets authorization privileges for all other users and secures the integrity of the system. By limiting full access ability to the DBA a tighter control can be kept on the system.

In addition to granting access privileges the DBA controls other aspects of the system. Any change to the system's design can only be made when authorized by the DBA who will first evaluate the advantages and disadvantages of any possible change. This implies that



COMPONENTS OF A DATA BASE SYSTEM

Figure 1

the DBA must constantly monitor and note any inefficient aspects of the system. The DBA alone is responsible for maintaining the integrity and effectiveness of the system. Centralizing control in this way limits the chances of unauthorized data manipulation.

The next level of users include those who are authorized to actually modify, insert or delete data in the data base. Each user in this group may be authorized to modify only a specified portion of the data base. For example, in the manufacturing company example, someone with expertise in the warehouse would be authorized to maintain the data pertaining to that department but would not be authorized to modify data pertaining to any other department. In addition these users might even be prohibited from accessing certain data (such as employee salaries).

The last and probably largest group of users would be those who are only authorized to access data. As with the previous level these users would most likely be authorized to access only a limited portion of the data base. Using this data, queries could be answered and reports generated.

By structuring the levels of users in this way data integrity can be maintained and security controlled. The DBA authorizes access and is responsible for periodically checking that standards are enforced.

Data

While there is only one actual set of data in the computer at any point in time, there are several ways in which this data can be viewed. One of the most important advantages of using a data base system is that the user can conceptualize the organization of the data in quite a different way from the actual data structures used. In this way a user inexperienced with computers can query the data and the relationships among data items without knowing the specifics of the data files accessed. A straightforward example of this concept would be to conceptualize an indexed sequential file of names as an alphabetized list of these names.

The overall user's view of the data is the conceptual or logical data base. The actual file structure used is the physical data base. Depending on the needs of the overall system the physical files could be trees, hashed files, sequential files or any appropriate combination of data structures. Only the DBA and a small group of systems programmers need be familiar with this aspect of the data. The types of logical and physical structures which can be used will be discussed in Chapters 3 and 4 respectively. The end user usually remains oblivious to the physical structure by using software which maps the actual physical

structure to the logical view the user sees. This is analogous to a programmer making calls to a read of an indexed file without actually having to be aware of the methods of linking and indexing the particular operating system actually employs.

As was mentioned earlier (page 8) a single user may only be authorized to access a portion of the entire data base. Consequently a particular user may only have to be familiar with that subset of the logical data base. This user view describes the logical structure and relationships among the data items that the user is authorized to access.

Languages

A main objective of a data base system is to hide the actual physical structures of the files from the end user[7]. The user need only be familiar with the logical structure. This is accomplished throught the use of a query language. The query language consists of a series of calls to subroutines written in a host language such as FORTRAN or PL/I. To retrieve a particular record from a file with a specific attribute value the user need only make a call such as

SELECT FROM <filename> WHERE <condition>.

The subroutine called from this code will determine whether the read used will be keyed or sequential and

any other parameters needed by the system. Updates, deletions and insertions can be handled similarly.

The query language is generally designed to correspond as closely as possible to the logical structure of the data base. In this way the user need have minimal computing experience in order to successfully manipulate the data. Only a fixed set of commands, modelled from the logical structure, need be learned.

The DBA however may have needs that can not be addressed through the query language. Since this user will have a thorough understanding of the physical structure of the data base, the host language itself can be used to manipulate data. The DBA can decide for instance to expand a two byte integer field to four bytes. A program written in the host language could be written to make this transformation. The DBA then will make the appropriate changes to the subroutines called by the query language which access or manipulate that field.

Another language used in a data base system is the data definition language or data dictionary. This is not a procedural language but rather a thorough description of the data structures both logical and physical[8].

On the logical level, this language should include

descriptions of all logical record definitions as well as the characteristics of each field within that record. An example is given in Figure 2.

```
EMPLOYEE_RECORD

Name:                        last,first
Social_Security_Number:      ddd-dd-dddd
Telephone_Number:            ddd/ddd-dddd
Salary:                      5000-85000
```

LOGICAL RECORD DEFINITION

Figure 2

Besides describing the logical records, the relationships that exist between records and data items should be listed in terms of the logical model of the data base. For instance if an employee's salary depended entirely on his rank which was also included somewhere in the data base this relationship should be clearly defined. Other information pertinent to the specific logical design chosen must also be included.

On the physical level each file structure should be defined. This should include indexing methods as well as storage techniques. For instance, a file which is accessed infrequently could be stored on a slower device then a frequently accessed file. A thorough description

of each physical record type should be listed. This record definition could differ considerably from any logical record since the fields in a logical record could indeed reside in more than one physical file. The descriptions of each field in the record should address the actual physical data structures used.

The data dictionary should be maintained in such a way that it can be easily accessed. Of course only a small group of users need address the physical description. The DBA alone should be authorized to make changes to the data dictionary. This readily available description of the data base assists in thoroughly understanding the totality of the system and thus allows the user to more readily extract information from the data base using the query language[9].

ADVANTAGES

There are several advantages to implementing a data base system as the means of maintaining an enterprise's information. During the discussion of the definition of a data base several positive ramifications were mentioned. The following summarizes these advantages.

Data Integrity Maintained

Whenever a user accesses any information collection it is with the belief that the information that will be retrieved is correct. If indeed the data is found to be faulty the reliability of any reporting based on that data is jeopardized. Consequently the value of any data collection is based on the correctness of that data. Several aspects of a data base system help insure that data integrity is maintained.

A main objective of a data base system is to minimize and control any redundant data the data base contains[10]. As has been discussed (page 6) storage of uncontrolled redundant information opens the system up to the possibiity of containing inconsistent information. This could easily lead to incorrect or questionable results. Since the data base is essentially controlled by the DBA, this user can easily insure in other ways that data integrity is maintained. Periodic spot checks of data fields can be made as well as checks that redundant data items are being modified consistently. The DBA can also insure that only qualified personnel are authorized to modify the data[11].

Security and Privacy Maintained

Some of the methods used to insure data integrity can also be used to enforce data security. The privilege to access any data field must be granted to only those users who have company authorization to see that data. Confidential information can therefore be protected since the DBA will insure that authorizations are correct. Periodic checks by the DBA can be made to verify that these restrictions are being enforced. The ease of maintaining the security of the data is the direct result of having centralized control of the data base system[12].

More Information Readily Available

An enterprise that does not employ a data base system to store its information limits the amount of information that can be extracted from its system. Reporting that relies on information stored by separate departments is cumbersome to generate. At best extra programming time would be required and might therefore be cost prohibitive. A ramification of implementing a data base system is that the entire enterprise's data is shared by all personnel in the enterprise[13]. A fixed

method of representing data and relationships among data is known. This in turn allows all users the ability to access data pertinent to several aspects of the company. Valuable information can be therefore more quickly retrieved.

Another aspect of a data base system that allows easy access to information is the query language[14]. This fixed set of commands are generally all the inexperienced user need become familiar with before being able to extract information from the system. The time consuming, costly, programming process is minimized. A user with a question can quickly query the system and extract the answer themselves.

While an enterprise might have had the same data stored before implementing a data base system that data would most likely be stored in ways that would make it virtually impossible to extract the volumes of information that the data provides.

Easily Adaptable

It is highly unlikely that any system design will remain static throughout an enterprise's existence[15]. Additions to the system will most definitely be made.

In the manufacturing company example, for instance, the company might decide that it would be helpful to include the current cost of a part.

Since the data base system is centrally controlled the addition of this new field can be made efficiently and correctly. The DBA can evaluate how a new field will fit into both the logical and physical data structures and make the necessary changes to both. The data manipulation and data definition languages can then be updated. The user need only be informed that this new information is accessible. The many existing application programs based on the query language can continue to work without change.

A modification to the data base design can be made without affecting the users' ability to access the data available[16].

REFERENCES

[1]Martin, James Computer Data Base Organization, 2d ed., (Englewood Cliffs, N.J.: Prentice-Hall, 1977) p. 22.

[2]Ibid., p. 24.

[3]Date, C. J. An Introduction to Database Systems, 3d ed., (Reading, Mass.: Addison-Wesley, 1982) p. 5.

[4]Martin, p. 37.

[5]Date, p. 13

[6]Martin, p. 43.

[7]Ullman, Jeffrey D. Principles of Database Systems, 2d ed. (Rockville, Maryland: Computer Science Press, 1982) p. 3.

[8]Ibid., p.10.

[9]Martin,James An End-User's Guide to Data Base, (Englewood Cliffs, N.J.: Prentice-Hall, 1981) p. 113.

[10]Date, p.10.

[11]Ullman, p.3.

[12]Date, p. 11.

[13]Ibid., p. 5.

[14]Martin, Computer Data Base Organization, p. 43.

[15]Ibid., p.29.

[16]Ullman, p. 9.

CHAPTER 3

LOGICAL VIEW OF A DATA BASE

In order to use any data base the user must be familiar with the logical structure of the data involved. This affects the way the user will update, modify or extract information from the data base. Of course the actual physical structure of the data may differ considerably from what the user perceives the structure to be but this difference should be disguised.

There are several logical approaches to data base design. The three most common approaches are:

1. hierarchical,
2. network, and
3. relational.

In order to illustrate the design concepts involved with these approaches a library data base will be considered.

Suppose a library wishes to maintain a data base which contains information about their current selection of books. Information about authors such as names and

addresses might be included as well as information pertaining to books such as the title, publisher, year published, number of pages and catalog number (assumed to be unique for each book). Also the relationships between data items should be maintained such as which books were written by a specified author.

HIERARCHICAL

Using the hierarchical approach the user views the data as a series of tree structures[17]. Figure 3 shows the library information in a hierarchical structure. In this case each tree has as its root a record describing the author. This parent record has as its children a list of records representing the books that author has written. There might be several children or none at all (as is the case of the author Patrick Powell). In general there can be any number of dependent records to any level. In any case however no dependent record can exist without a superior. So in this example a book without an author could not be listed without creating a dummy author record.

In the example given (Figure 3) each record uniquely describes a real world entity, either an author or a book. This is not always the case in hierarchical structures. For example, a dependent record could be

Jones, John	402 Park Ave., N.Y., N.Y.

Computer Are Fun	760.4	1982	542	Wiley

Smith, William	57 Willow St., Trenton, N.J.

Data Base Design	759.7	1979	124	Sribner's
User Interface	760.93	1983	311	Wiley

Adams, Robert	Box 43, Des Moines, Iowa

FORTRAN	760.82	1981	421	Little Brown
FORTRAN II	760.83	1984	184	Brooks

Brown, Mary	University of Maine, Orono, Maine

Programming in BASIC	760.71	1981	256	IBM
FORTRAN II	760.83	1984	184	Brooks

Powell, Patrick	72 Parkway, Little Falls, N.J.

HIERARCHICAL VIEW OF LIBRARY DATA

Figure 3

created for books which provides additional information about the book such as the section of the library in which it is found. In this case then this new record is providing information about a book and not uniquely identifying it. In particular that record would not have any meaning without being viewed with its superior.

Relationships between entities are provided through links[18]. To know for instance what books were written by William Smith one would have to follow the link between the author record and the list of book records. Two constructs, are therefore needed, records and links. The user must be familiar with both.

In order to maintain the data base the user must be able to update data, delete data or insert data into the structures provided. In this case the user must be able to detect the type of record being accessed. Suppose the library acquired a new book by William Smith. After creating the record containing the information about the new book a search would be made for William Smith's author record. The book record then would be placed in the corresponding list of book records. To delete a book by William Smith not only must the appropriate book record be deleted but a check must be made that the link from the author record to the book list is still correct. It is clear that the various types of records

used as well as the links make these operations complex and, hence, prone to error[19].

Now consider now some possible queries on this data base and the algorithms that would solve these queries.

Query 1: Find the titles of books written by William Smith.

```
Repeat
   Read next author record
      If author is William Smith then
         While more books in this author's
   list
            Read book record
            Print book title
         end
Until William Smith author record found
```

Query 2: Find the authors' names of all books with more than 200 pages.

```
While more records
   Read next author record
      While more books in this authors list
         Read book record
         If pages in book > 200 then
            Print author's name
      End
End
```

Note how the user must be concerned with several issues not inherent in the query itself[20]. For instance the type of record being accessed must be determined and the notion of lists of books under an author record must be understood. The users must have a solid understanding of the structures involved in order to query the data base correctly.

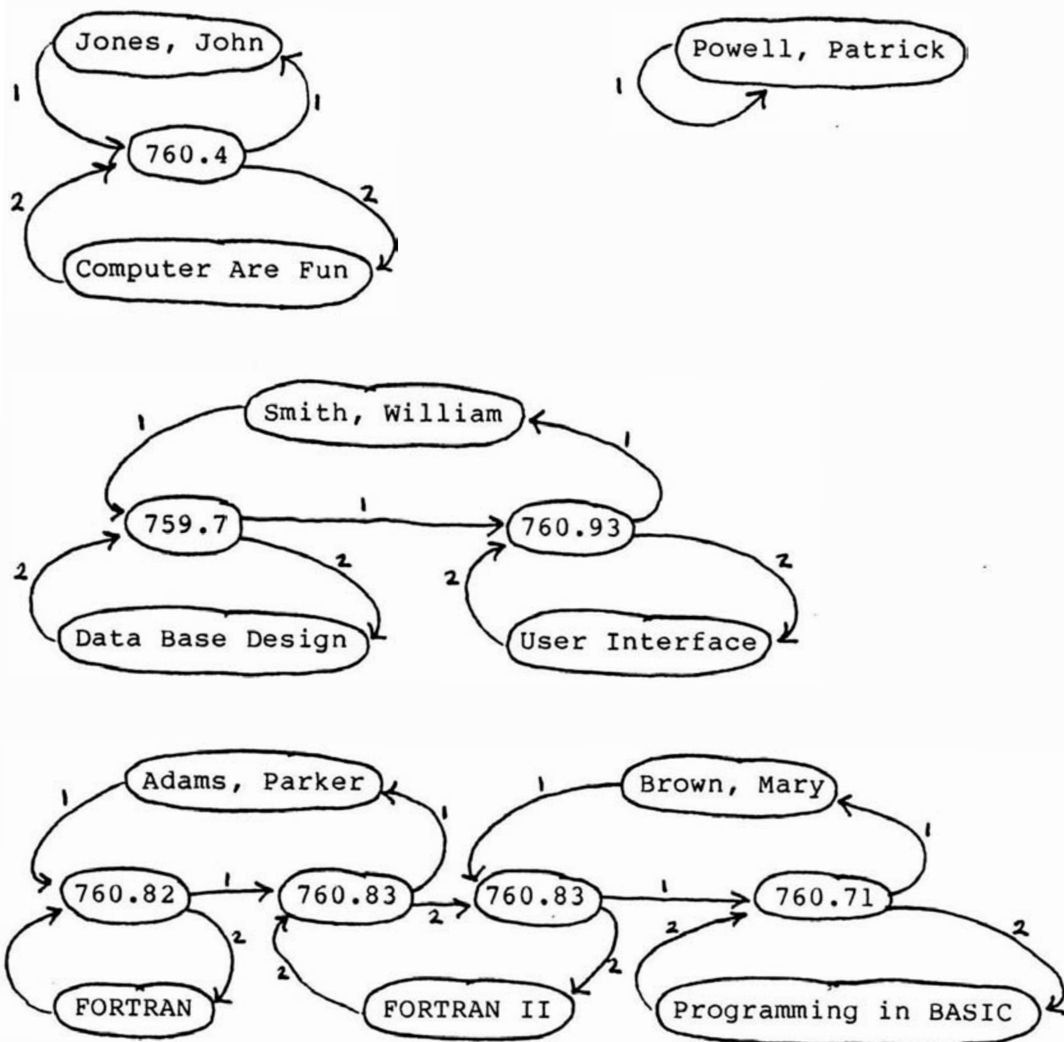
Recall now that one of the main objectives of a data base system was to minimize redundant data. Consider then the book FORTRAN II which has two authors. Two equivalent instances of the book record then are included in the data base. This is the result of having a many to many relationship between the root and child records, i.e. one book may have many authors and one authors could have written several books. Clearly this will be not be an uncommon occurrence and therefore opens the system up to the problems associated with redundant data. This is obviously undesirable and the design should therefore be modified to avoid this problem. There are methods available for implementing many to many relations in a hierarchical structure[21] but they generally complicate the user's view of the data. A network design supports this type of relationship more easily[22].

NETWORK

The user can now view the data on a series of chains linking the records in various ways[23]. In addition to author and book records a third type of record, a connector, is also used. The diagram in Figure 4 illustrates (using shortened versions of each record) the users view of the library data base under a network approach.

In this example the connector record represents the association between authors and books and gives information pertaining to this association[24]. All connectors for a given author are placed on a chain (labeled 1) beginning and ending at the author record. Similarly all book records initiate and end a chain (labeled 2) containing its corresponding connector. A connector then in this example lies on at most two chains. If an additional record was included pertaining to each book a third chain could be initiated.

Note in this approach there are no superior or dependent records. Consequently the insertion of a book without an author does not necessitate the creation of a dummy author record. Rather the book record will chain with its connector but the chain to an author will be empty.



NETWORK VIEW OF LIBRARY DATA

Figure 4

As with the hierarchical design a real world entity may not necessarily be defined by a record alone. The record and the information provided by and/or through the connector record may be needed.

Relationships between entities are determined through links[25]. Note now that the representation of the many to many relationship is no longer a problem[26]. A book that has several authors will merely lie on a chain that links all those authors through various connectors (see Figure 4).

Suppose the library wished to insert a new book by an author who is already listed. In addition to creating the appropriate book record care must be taken to adjust the links on the book chain corresponding to this author. If a book was withdrawn from circulation not only must the book record be deleted but two chains must be modified correctly. It is clear that the addition of more than one type of link complicates the insert and delete operations further[27].

Consider now the two sample queries.

Query 1: Find all titles of books written by William Smith

```
While more records
   Read author record
   If author is William Smith then
      Follow chain 1
      While books on chain 2
         Print title
      End
End
```

Query 2: Find the authors' names of all books with more than 200 pages.

```
While more records
   Read author record
      Follow chain 1
      While books on chain 2
         If pages > 200 then
            Print author's name
      End
End
```

Now the user must be concerned with not only the various types of records but the different types of chains which must be followed in order to correctly find the information desired[28]. The user is again confronted with

complexities which have nothing to do with the query posed.

RELATIONAL

In this case the data is viewed in the form of tables of information. The three tables in Figure 5 are an example of how the library data might be represented.

While the user can simply view the structures as tables of data, in reality these tables are mathematical relations and must therefore obey certain constraints[29]. Every value in a particular column (attribute) must be drawn from a predefined set of possible values (domain). For instance the values under the attribute CAT_NUM must belong to the set of all valid catalog numbers. Every row (tuple) must be different and must therefore be uniquely determined by a set of attributes (key). For instance each tuple in INFO is uniquely determined by NAME and ADDRESS.

Each tuple in these relations describes uniquely a real world entity[30] in the library data collection, either an author or a book. Relationships between authors and books are represented by values under given attribute names[31]. For instance the fact that William Smith has authored two books in the library is represented by the fact that his name appears twice

AUTHORS

NAME	TITLE	CAT_NUM
Jones, John	Computer Are Fun	760.40
Smith, William	Data Base Design	759.70
Smith, William	User Interface	760.93
Brown, Mary	Programming in BASIC	760.71
Adams, Robert	FORTRAN	760.82
Adams, Robert	FORTRAN II	760.83
Brown, Mary	FORTRAN II	760.83

INFO

NAME	ADDRESS
Jones, John	402 Park Ave.,N.Y.,N.Y.
Smith, William	57 Willow St., Trenton, N.J.
Brown, Mary	University of Maine, Orono, Maine
Adams, Robert	Box 43, Des Moines, Iowa
Powell, Patrick	72 Parkway, Little Falls, N.J.

BOOKS

CAT_NUM	YEAR	PAGES	PUBLISHER
760.40	1982	542	Wiley
759.70	1979	124	Scribner's
760.93	1983	311	Wiley
760.71	1981	256	IBM
760.82	1981	421	Little, Brown
760.83	1984	184	Brooks/Cole

RELATIONAL VIEW OF LIBRARY DATA

Figure 5

under the attribute NAMES in AUTHORS. Similarly it is clear that John Jones wrote a book which was published in 1982 since the catalog numbers are identical in the two tuple in AUTHORS and BOOKS. Entities and the relationships between entities then can be represented in these tables or relations with no other data structure necessary.

The operations used to modify the data base are now straightforward. For instance when a new book is brought into the library three new tuples can be inserted into the three relations by simply providing the information needed for the three records and appending these new records onto the tables. Of course a check must be made to insure that the author tuple is not already listed in the AUTHOR table since each tuple must be unique. In order to delete a book from the data base the records involving that book must be located in the tables and deleted but the record involving the author's address could be maintained for further reference.

In order to answer queries posed the user simply manipulates the tables of data.

Query 1: Find the titles of books written by William Smith.

```
While not end of table AUTHORS
   Read next NAME in AUTHORS
   If NAME = Smith, William then
      Print title
End
```

Query 2: Find the authors' names of all books with more than 200 pages.

```
While not end of table BOOKS
   Read next PAGES in BOOKS
   If PAGES > 200 then
      Num = CAT_NUM
      While CAT_NUM    Num
         Read next CAT_NUM in table AUTHORS
      end
      Print NAME in AUTHORS
End
```

When using relational data bases the data manipulation language will often provide the user with several higher level operators which can be used to answer queries with a simple set of commands[32]. These operators are based on the theory of relational algebra but still allow the user to view the procedures as simple table manipulations.

ADVANTAGES OF THE RELATIONAL MODEL

In the last section three logical data base models were presented. Of the three the relational model has several major advantages.

Understandability

Perhaps the most important advantage of the relational model is its ability to be understood by all levels of users. Clearly this should be a primary objective when choosing a model since the data base system will be of little value if the users can not access it without difficulty[33].

One characteristic of the relational model which makes it understandable is the fact that it uses only one construct to represent all the information in the data base[34]. The relation, which can be viewed simply as a table of data, is familiar to all levels of users. Any row (tuple) in the table clearly identifies a unique entity in the enterprise. Relationships between entities are represented by the listing of equal attribute values in more than one table.

This single construct contrasts with the hierarchical and network models which rely on various types of records and links to represent the same

information. A real world entity in this case is shown through the linking of several records, each record only providing some information about the entity and not uniquely identifying it. In order to see the relationships between entities other links may have to be followed. Clearly these are more complex models in which the user must be familiar with several types of constructs and how they fit together in order to view the data base in its entirety.

The relational model has retreated from the complexities of the hierarchical and network models and hence provides a more easily understood picture of the enterprise's information[35].

Ease of Manipulation

Not only should the data base design be understandable to all levels of users it should be constructed in such a way that it can be easily manipulated. Users should be able to retrieve, update, insert and delete data without unnecessary complexity.

Because the relational model depends on only one construct it is easier to implement data manipulation procedures[36]. For instance, in order to update information in the relational data base it is simply

a matter of searching for the appropriate tuple in the corresponding files, allowing the user to update it and then rewriting it to its location. In the cases of the network and hierarchical models not only must the appropriate record be found and updated but it must be determined that the links involved with that record are still correct. Clearly this procedure can become quite complex as the model itself becomes more complex. Deleting a record in the network and hierarchical models also requires careful updating of links. In fact the deletion of certain types of records could imply the deletion of several of its dependent records. Consequently while the relational model basically needs only one type of update, delete and insert the network and hierarchical models will need many more, each[37] dependent on the type of record being manipulated.

In terms of querying the data base the relational model has a set of higher level commands that allow the user to basically cut and past tables of information to construct new tables[38]. The fact that these relational operators are closed allows the user to reapply the operators to any result in order to retrieve more detailed information[39]. Generally the query languages used in the network and hierarchical models are more low level and require more operators to manipulate the

various constructs involve. Since in these models relationships depend on the linking of various record types the user is forced to consider navigational strategies in order to derive desired information. Often users are forced to spend an inordinate amount of time considering the strategy and implementing it resulting in less information being readily available. The added complexity also tends to inhibit unanticipated queries due to the extra time needed to derive the result.

Since the relational model's design lends itself to more straightforward manipulation techniques the typical user will find it easier to access and retrieve information from a relational data base[40].

Theoretical Basis

Generally there is no known theoretical support for the design of either the hierarchical or network models. The relational model however is supported by two theoretical foundations[41].

The single construct of the relational model is actually the mathematically defined relation. A relation is a subset of a cross product of two or more sets. The set of relational operators manipulates these sets and yields new sets.

Consequently the relational operators are defined in terms of mathematical set theory operators. The fact that a relation is a known mathematical construct implies that its behavior will be more predictable and more likely to appeal to the user's intuition.

The process of designing a relational data base is supported by another theoretical base. Normalization theory provides a set of rigorous guidelines for the design of the logical model. In this way the designer of the data base can insure that potential pitfalls and problems in design can be identified and possibly eliminated.

The theoretical bases for the relational model help guard against unpredictable problems in the future use of the data base system.

Adaptable

It would be rare for a data base design to remain static over the course of its existence. As an enterprise expands and changes adjustments will most likely have to be made to the data base's logical design to reflect these changes. One objective of a data base system then is that these modifications can be made with as little disruption as possible to the currently running system[42]. Thus the notion of data independence is supported.

The simple design of the relational model provides a basis for independence between application programs and the physical structure of the data[43]. In general when accessing a network or hierarchical modelled data base the application programmer must consider and navigate the various links involved. If say a new record type and its links are introduced to these systems many of the existing application programs that rely on the old structure will have to be updated. This is because the new navigation path may be different from the old due to the addition of the new record and its links. The relational model's application programs do not rely on navigational techniques. If a new relation is added to the system the existing application programs can continue to run without any change. If a new attribute is added to an existing relation the existing application programs may (or may not) have to be updated to reflect this new field in the known record but with only that modification continue to run. No change to the program's logic will be needed. Generally the only time existing application programs in a relational system will no longer run is when an attribute that the application program depended on is deleted.

In general the relational system is more flexible and therefore more capable of adapting to the existing environment. In fact it was this issue that Codd

primarily addressed in his 1970 paper in which he first introduced the idea of a relational data base.[44]

The relational model has several important advantages that tend to make it a more desirable system. One disadvantage that has been cited deals with the inefficiencies that arise when using some of the relational operators on large data bases. It has been noted for instance that careless use of the cross product operator in conjunction with other operators could lead to large amounts of useless data being generated. This in turn wastes storage space and time. In small data bases this may not be a serious problem at all[45]. In large data bases the concept of optimizing a set of relational commands has been used resulting in programs as efficient as those in any other data base system[46]. Therefore the relational model with its simplicity tends to be the preferable design.

REFERENCES

[17]Date, D. J., An Introduction to Database Systems, 3d ed., (Reading, Mass.: Addison-Wesley, 1982) p. 67.

[18]Ibid. p. 68.

[19]Ibid. p. 69.

[20]Ibid. p. 68.

[21]Ullman, Jeffrey D., Principles of Database Systems, 2d ed. (Rockville, Maryland: Computer Science Press, 1982) p. 126.

[22]Date, p. 70.

[23]Ibid.

[24]Ibid.

[25]Ibid. p. 78.

[26]Ibid.

[27]Ibid. p. 73.

[28]Ibid.

[29]Ibid. p. 65.

[30]Ullman, p. 21.

[31]Date, p. 65.

[32]Date, p. 77.

[33]Ullman, p. 168.

[34]Date, p. 478.

[35]Ullman, p. 170.

[36]Date, p. 478.

[37]Ibid. ,p. 73.

[38]Ibid., p. 478.

[39]Ibid,. p. 479.

[40]Ullman, p. 168.

[41]Date, p. 479.

[42]Martin, p. 39.

[43]Ibid., p. 226.

[44]Codd, E. F., "A Relational Model of Data for Large Shared Data Banks", Communications of ACM, (Vol. 13, #6, June, 1970) p. 387.

[45]Martin, James, Computer Data Base Organization, 2d ed., (Englewood Cliffs, N.J.: Prentice-Hall, 1977) p. 225.

[46]Ullman, p. 170.

CHAPTER 4

PHYSICAL DESIGN OF A DATA BASE

While the major thrust of this paper is on the logical aspect of designing a data base some attention to the physical design is helpful in order to get an understanding of the entire data base design process.

As discussed in Chapter 2 the Data Manipulation Language (DML) will hide from the user the actual physical data structures used. The developer of the DML can view the physical data as a set of files. Each file contains all occurences of one type of physical record. For each file the DML must know the record defintion, the fields in each record, any sequencing existing in the file and (if direct accessing is available) what keys can be used. The DML can take advantage of any accessing methods the particular operating system provides.

When choosing the type of file structures to use several aspects of the entire data base system and its

environment must be considered. The following summarizes some of these considerations.

STORAGE SPACE AVAILABLE

If the enterprise's computer system is such that high speed memory is limited, the designer of the data base must insure that each file is structured to take best advantage of any space used.

There are several methods which can be used to conserve space in each file. Consider for instance a file which includes a 30 character representation of the department an employee works in. Since in general one company will have a small fixed number of departments and many employees working for each department a unique integer can be used to represent each department. This integer, taking only 1 word, can then serve as an index into a small file which actually lists the character descriptions of the departments[47]. Of course any listing that required the actual department name would require at least one extra seek but the saving of space balanced against this extra time might be preferred.

A similiar method that can be used to conserve space is a 2-dimensional bit map in which a 1 in a particular i,j position determines a field value for the record associated with the ith row. This method is

particularly useful for representing variable length records[48]. Consider, for instance, a record containing a student's course schedule. Rather than maintain variable records for each student containing course numbers a bit map could be used where each row of the table represents a unique student and a 1 in a specified column implies that student is currently enrolled in that class.

When space is severely limited the data base designer may decide to store infrequently used files on slower mediums such as magnetic tape[49]. This would cause a serious delay in accessing that data but might prove necessary.

VOLATILITY OF A FILE

While the data base designer can expect that almost all of the files will be subject to deletions and insertions some files will be more volatile than others. The fact that a file will be highly volatile should be taken into consideration when choosing a file structure. It is known, for instance, that tree structures become unbalanced after several insertions and deletions. This in turn effects the efficiency of searching on this structure. A similiar argument can be made concerning

hash files where after several insertions the rate of collisions will increase. If the data base designer still prefers these structures periodic reorganizations of the files can be made during times the system is not active.

In general, however, some structures should be avoided for files where the rate of deletions and insertions is high. Suppose the designer of the data base ordered the records in a sequential file by some field value. Then each time a new record is added to the file several records may have to be physically moved in order to maintain the file's order. This could prove very time consumming. Some techniques which could be used instead include ordering the file by the time of insertion of the records or using pointers to maintain a file sequence[50].

In systems where the rate of change is extremely high it might be advantageous to do no real time inserting and deleting but rather maintain a separate file of the records to insert and mark those to be deleted[51]. This differential file of records to be inserted can be designed to be accessbile for searching if the desired record is not found in the permanent file. During a time when the system is not highly active the differential file can be merged into the actual file system and marked records deleted.

SPEED OF RETRIEVAL

When the designer of the data base is evaluating various data structures a major consideration should be the speed at which records can be retrieved, especially if queries are to be processed interactively. If the value of the system depends on the ability to quickly retrieve records, this should affect the designer's choice of both the data structures and the storage devices. Also taken into consideration should be any delay in response due to the communication system. For instance, if using telephone lines causes a delay in response the data base system should compensate as much as possible.

Suppose the data base was to be used for an airline's reservation system. Obviously any delay in response time here would be undesirable. After the data base designer has obtained the hardware that best meets this goal, several data structures and data base techniques can be used to also insure that the response time is fast.

In general it is best to design the system so that all accesses are to single records via their primary key[52]. Searching should be avoided if possible, but if necessary designed so that it can be accomplished quickly perhaps via indices in main memory. The designer of the data base can decide which information

will be queried most frequently and store it in the fastest medium. Some operations such as inserting and physically deleting may be delayed until activity on the system is slow. In some cases the data base designer might decide to maintain a condensed copy of the data which is accessed most frequently. This smaller set of data can be stored on the fastest medium but requires careful periodic reconciliation of information[53]. This is a case where the data base design objective of eliminating redundant data is waived but controlled.

In many cases fast response time is needed to provide a friendly dialogue between the computer and the user. It would be frustrating for a user, who say has a customer on the telephone, to have to wait for half a minute to get the information requested. The data base designer can try to develop the dialogue in such a way so that delays will occur at points when the user is prepared to relax for an instant[54]. Suppose the user is going to update a record. While the sytem is retrieving the record the user can be typing in information pertaining to the update. In this way, the delay in searching for the record is hidden. After the user has entered all the updated information the user is generally prepared to relax for a moment. At this point then any delay caused by say rewriting the record to the file would not neccessarily be undesirable. Thus, by

interleaving appropriate commands the data base designer can take advantage of the user interface when developing queries.

MULTIPLE INDEXES

In many systems it is desirable to be able to retrieve records quickly via many fields as opposed to strictly via the primary key. Since the primary key usually determines the physical address of the record some other method must be used to quickly access records with a specific value in a given field. Additional records may also have the same value in that field since only the primary key is known to be unique in each record.

One strategy that can be used is to build an index for each secondary field that will be accessed[55]. This generally implies using pointers to the records that satisfy a specific field value. Variable length records would be needed in the index file since the number of pointers needed will be unknown. Another, perhaps more satisfactory, technique would be to have the index point to a chain of records with that common attribute[56]. In this case the actual data record must contain a pointer field for each field that might be accessed this way, but the number needed would have to be known in advance.

Of course the use of any index file requires additional storage space and maintenance. Consequently, a tradeoff is made between the ease of accessing a record via these fields and the extra space and programming required. The data base designer must carefully examine the options before committing the system to a particular structure.

THE LOGICAL DESIGN

Besides maintaining the data items themselves the physical level must also represent in some way the relationships between data items. Recall from the previous section that the three logical appproaches represent the relationships in different ways: the network model uses links through connector records, the hierarchical uses links between types of records and the relational uses the listing of equal attribute values from common domains in different relations. The DML must transform the physcial representation into the logical and insure that the user can properly take advantage of the logical relationships[57].

Various pointer techniques can be used to represent the relationships in the network and hierarchical models. Of course, the more complex the logical design the more complex the physical representation will be to

create and maintain. In general the simplest logical model to implement is the normalized relational model where links to represent relations are not necessary[58].

The previous discussion listed several important issues which the data base designer must consider when choosing the physical data structure to represent a particular logical model. However, tradeoffs in efficiency occur between different data structures. The following summarizes some of these tradeoffs.

PHYSICAL DESIGN TRADEOFFS[59]

Table 1

EMPHASIS ON	TENDS TO
Speed of retrieval	Increase storage space
Speed of retrieval	Increase cost of hardware
Flexibiility of searching	Decrease speed of retrieval
Flexibility of searching	Increase storage space
Minimize storage space	Complicate maintenance
Noncomplex structures	Increase retrieval speed
Complex structures	Complicate recoverability

The data base designer must decide what the system's priorities are and work from there. Tradeoffs will then be made with these priorities in mind.

REFERENCES

[47]Martin, James, Computer Data Base Organization, (Englewood Cliffs, N.J.: Prentice-Hall, 1977) p. 575.

[48]Ibid. p. 329.

[49]Ibid. p. 314.

[50]Ibid. p. 619.

[51]Ibid. p. 620.

[52]Ibid. p. 631.

[53]Ibid. p. 636.

[54]Ibid. P. 635.

[55]Ibid. p. 465.

[56]Ibid. p. 468.

[57]Date, C. J., An Introduction to Database Systems, 3d ed., (Reading, Mass: Addison-Wesley, 1982) p. 63.

[58]Martin, p. 316.

[59]Ibid. p. 309.

CHAPTER 5

THE RELATIONAL DATA BASE MODEL

There are essentially two components to the relational data base model, the relational constructs used to contain the data and their relationships and the relational algebra used to manipulate the data.

RELATIONAL CONSTRUCTS

As shown in Chapter 3 the foundation for the definition of a relational data base is simply the concept of a table of information. This logical approach to data manipulation is especially convenient for an inexperienced user since the table is a common way to list and view data. In order to insure the data can be correctly retrieved using the data manipulation language certain guidelines must be placed on the construction of the tables. Using the mathematical

theory of relations a relational data base can be defined.

Consider the following example. A personnel office of a small company wishes to maintain data concerning their employees. This could include their names, addresses, telephone numbers, departments they work for, dependents and salaries. One table might be constructed as in Figure 6.

CURRENT_EMPLOYEES

NAME	STREET	CITY	ST	TELEPHONE
John Jones	23 Main Street	Richmond	Va.	355-1234
William Smith	55 Walnut Road	Richmond	Va.	257-2343
Mary Brown	123 South Street	Goochland	Va.	223-1234
Ann Williams	Box 1422	Richmond	Va.	446-1233
Robert Brown	84 Conover Street	Richmond	Va.	678-4433
Paul Jones	47 Wayside	Ashland	Va.	432-7895
Robert Field	99 Boulevard	Richmond	Va.	789-5656
Howard Evans	Box 34	Goochland	Va.	987-8989

CURRENT_EMPLOYEES RELATION

Figure 6

Domains And Attributes

Each column in a table contains one type of data. The values that appear in a column then can be considered members of a domain that defines that data type. For example in Figure 6 the column of names could be considered a subset of D_1, the set of character

strings of, say, length 30. Each column's domain could be defined in a similiar way.

DEFINITION: A domain is a nonempty fixed set of data items that are of the same type.

In Figure 6 it is clear that the values in the column NAME are a specific use of the domain, character strings of length 30, from which the values are drawn. NAME, then, which distinguishes that use is considered an attribute.

DEFINTION: An attribute value is the specific use of a domain value in a relation.

The same domain could be used in several ways in the database. For instance names of retired employees could be represented in RET_EMP with their values elements of the same domain used by NAME. Each distinct use of the domain could be assigned a different attribute name distinguishing the way the values are used.

Tuples

Examining the rows of the table in Figure 6 it is clear that each row describes a different employee or a different actual entity. The attribute values in that row describe or provide information about the entity. A

row is generally referred to as a tuple and is analagous to a record in a file. Since there are five attribute values in each row of the table in Figure 6 the tuples are referred to as 5-tuples.

DEFINITION: An n-tuple is a set of n attribute values describing an entity.

Relations

The table in Figure 6 then is simply a set of 5-tuples each of whose values are drawn from specified domains and listed in a specific order. This table then satisfies the defintion of a relation.

DEFINITION: Given a collection of Domains $D_1, D_2, \ldots, D_n$, R is a relation of degree n on those n domains if it is a set of ordered n-tuples $<d_1, d_2, \ldots, d_n>$ such that $d_1 \in D_1$, $d_2 \in D_2$, ..., $d_n \in D_n$.

A relation can be denoted by $R(A_1:D_1, A_2:D_2, \ldots, A_n:D_n)$ where A_i is the attribute name for the ith column whose values are drawn from the ith domain. When the domains are defined in context the shorter notation $R(A_1, A_2, \ldots, A_n)$ may be used. The relation in Figure 6 can then be denoted as

CURRENT_EMPLOYEES(NAME, STREET, CITY, ST, TELEPHONE).

Keys

Consider the information that is contained in CURRENT_EMPLOYEES. Clearly it would be of no use to list an employee more than once. Given that each tuple is unique it is apparent, then, that the combination of the five attributes uniquely determines each tuple in the relation. A proper subset of attributes however is often all that is needed to uniquely determine the tuple. Assuming (prehaps unrealistically) that no two employees will have the same name in CURRENT_EMPLOYEES, NAME satisfies this property and is therefore considered the key of the relation. This is denoted by CURRENT_EMLOYEES(<u>NAME</u>, STREET, CITY, ST, TELEPHONE).

DEFINITION: A key of a relation R is the smallest set of attributes that uniquely identifies each tuple in R.

In some cases more than one set of attributes might qualify to be the key of a relation. As another example consider the relation LOCATION (Figure 7) describing the departments employees are qualified to work in. Note a particular employee might be qualified to work in more than one department. Therefore the name alone does not

qualify as being the key for this relation. However the combinations of attributes (NAME, DEPT) and (NAME, DEPT#) uniquely determine each tuple. Both of these sets of attributes are considered candidate keys and one will be designated as the primary key.

LOCATION

[NAME, DEPARTMENT, DEPT#]

NAME	DEPARTMENT	DEPT#
John Jones	Engineering	423
William Smith	Development	472
Mary Brown	Resources	415
Ann Williams	Personnel	402
Robert Brown	Accounting	410
Mary Brown	Accounting	410
John Jones	Development	472
Paul Jones	Engineering	423
Robert Field	Maintanence	411
Howard Evans	Personnel	402
William Smith	Resources	415

LOCATION RELATION

Figure 7

Suppose that an employee moves and hence changes their address. The information listed in CURRENT_EMPLOYEES (Figure 6) must be updated to reflect this change. It is imperative that the exact location of the appropriate tuple be known before an update can be completed. Uniqueness is important to insure that the correct tuple is modified. The use of the primary key insures that the update is completed correctly and

therefore must be specified clearly for each relation defined in the data base.

Consider the relation PROMOTIONS (Figure 8) in which the personnel office wishes to maintain a list of employee's promotions and the date they occurred. Note that the promotion date was not known when the personnel office entered the information that Paul Jones was promoted to Senior Engineer. Therefore the date value might be null (denoted by ?) for a given tuple. In this relation then the date plays no role in uniquely identifing a tuple. Since it does not provide any distinguishing information about the employee in question it must not be considered an element of the key. Note also that a given name value could appear more than once in the relation if an employee has had several promotions. The key of the relation then must be the set (NAME, NEW_POSITION). In general an attribute that may assume null values should not be considered an element of the key[60].

PROMOTIONS

[NAME, NEW_POSITIONS, DATE]

NAME	NEW_POSITIONS	DATE
John Jones	Manager	9/15/83
William Smith	Senior Programmer	1/ 4/82
Ann Williams	Director	12/ 3/83
Paul Jones	Senior Engineer	?
Robert Field	Supervisor	6/ 7/84
Mary Brown	Systems Analyst	7/31/82
John Jones	Assistant Manager	4/23/78
Howard Evans	Researcher	5/26/82
Ann Williams	Systems Analyst	3/23/80

PROMOTIONS RELATION

Figure 8

First Normal Form

Consider the table of family information (Figure 9) in which the personnel office maintains information concerning the employees' dependents perhaps for insurance reasons. Note that some of the rows contain blanks since the person who constructed this table simply listed the name of the employee once if that employee has several dependents. While it can be argued that this table obeys the definition of a relation there are several problems inherent in its construction.

NAME	DEPENDENT	RELATION	AGE
John Jones	Louise	wife	32
	James	son	5
William Smith	Mary	wife	43
	Mary	daughter	15
	Mark	son	17
Mary Brown	Susan	daughter	3
	James	son	5
Robert Brown	Frances	wife	57
Howard Evans	Paul	son	14
	Robert	son	16
	John	son	10

FAMILY INFORMATION

Figure 9

Recalling that a key uniquely identifies a tuple in a relation it is clear that there is no combination of attributes that satisfies this property. Obviously the set of attributes (John Jones, Louise, wife) uniquely identifies the first row but the second row contains a blank value for name and there is more than one listing of (, James, son) in the table. This is the result of having a set of values, the dependents, associated with a single occurence of an employee's name. For example, the single listing of John Jones has associated with it a set containing two dependents. This is inconsistent with the defintion of the key of a relation and will cause problems with the the Data Manipulation Language's ability to uniquely find a record. To avoid this all relations should satisfy First Normal Form.

DEFINTION: A relation is in First Normal Form if each attribute value is atomic (single valued).

This means that the table FAMILY should be reconstructed without using sets of values for a given attribute. This is accomplished by simply listing the employee's name in each tuple (Figure 10). The key then is the set (NAME, DEPENDENT, RELATION) and the uniqueness of each tuple is insured.

FAMILY

[<u>NAME</u>, <u>DEPENDENT</u>, <u>RELATION</u>, AGE]

NAME	DEPENDENT	RELATION	AGE
John Jones	Louise	wife	32
John Jones	James	son	5
William Smith	Mary	wife	43
William Smith	Mary	daughter	15
William Smith	Mark	son	17
Mary Brown	Susan	daughter	3
Mary Brown	James	son	5
Robert Brown	Frances	wife	57
Howard Evans	Paul	son	14
Howard Evans	Robert	son	16
Howard Evans	John	son	10

FAMILY RELATION

Figure 10

The first component, then, of the relational data base model is a collection of relations that satisfy

First Normal Form. The user is confronted with what appears to be a series of tables of information. By constructing new relations using the relational operators, the second component of the relational model, the user can then extract a large amount of additional information.

RELATIONAL ALGEBRA

To extract information from a relational data base the user constructs new relations from those defined in the data base schema. A tool that can be used to do this is the set of operators defined in relational algebra. These operators manipulate one or more relations to obtain a new relation. The user can view these procedures as "pasting and cutting" tables of information in order to derive new tables.

Select

Consider the relation EDUCATION (Figure 11) and suppose a user wanted a listing of all employees who

EDUCATION

[NAME, DEGREE, MAJOR]

NAME	DEGREE	MAJOR
John Jones	MS	Engineering
William Smith	MS	Computer Science
Mary Brown	BS	Computer Science
Ann Williams	BA	Business
Robert Brown	MA	Business
Paul Jones	MS	Computer Science
Howard Evans	BA	Business

EDUCATION RELATION

Figure 11

have degrees in Computer Science. The SELECT operator will choose that horizontal subset of tuples where the value of the attribute MAJOR matches the value "Computer Science". This operation can be denoted by

```
PROG [NAME, DEGREE, MAJOR]:=
          EDUCATION WHERE MAJOR = 'Computer Science'
```

and the result is listed in Figure 12.

PROG

NAME	DEGREE	MAJOR
William Smith	MS	Computer Science
Mary Brown	BS	Computer Science
Paul Jones	MS	Computer Science

PROG RELATION

Figure 12

PROG is the name given to the resulting relation and its attribute names are given in the square brackets. Often these attribute names will be the same as in the operand relation(s) but this is not necessary. (The notation := is used to indicate an assignment is being made distinguishing this from the equality operator used for comparison).

SELECT determines a new relation R' which is a subset of tuples from a relation R. The tuples in R' are those that satisfy a boolean expression involving an attribute value from R and other values from the same domain. The resulting relation will have the same attributes as the original.

DEFINITION: Let $R(A_1:D_1,...,A_n:D_n)$ be a relation. Let op denote any one of the comparison operators =,<,<=,>,>=,<>. Let v_i be either a domain value from D_i or another attribute name defined on D_i. Let R' be the result of selecting from R where a_i op v_i. Then $R'(A_1:D_1,...,A_n:D_n)$ is given by the tuples $(a_1,...,a_n)$ where $(a_1,...,a_n) \in R'$ iff it appears in R and a_i op v_i evaluates to true.

As another example of SELECT consider the following. Suppose a relation was available which showed the date employees were hired, the salary they were hired at and their current salary (Figure 13). To find those employees who are currently earning the

HISTORY

[NAME, ST_PAY, CUR_PAY]

NAME	ST_PAY	CUR_PAY
John Jones	27000	40000
William Smith	28000	28000
Ann Williams	24000	32000
Howard Evans	22000	22000
Robert Field	32000	38500
Paul Jones	40000	40000

HISTORY RELATION

Figure 13

same amount as when they were hired one could use

```
SAME[NAME, ST_PAY, CUR_PAY]:=
                HISTORY WHERE ST_PAY = CUR_PAY.
```

In this example the equality test involves two attribute names as opposed to an attribute name and a fixed value. Every tuple is examined and when the values under these names are equal that tuple becomes an element of SAME.

Projection

Suppose a user desires a listing of all current employees and their telephone numbers. The user essentially wants to extract the columns listing NAME and TELEPHONE from the relation CURRENT_EMPLOYEES

(Figure 6). The projection operator takes that vertical subset and can be denoted by

NUMS [NAME, TELEPHONE]:=
CURRENT_EMPLOYEES [NAME, TELEPHONE].

The result is listed in Figure 14.

NUMS

NAME	TELEPHONE
John Jones	355-1234
William Smith	257-1234
Mary Brown	223-1234
Ann Williams	446-1233
Robert Brown	678-4433
Paul Jones	432-7895
Robert Field	789-5656
Howard Evans	987-8989

NUMS RELATION

Figure 14

Projection "cuts out" the specified columns from a relation and eliminates any resulting duplicate tuples.

DEFINITION: Let $R(A_1:D_1,\ldots.,A_n:D_n)$ be a relation and let $A' \subseteq \{A_k : A_k$ is an attribute of $R\}$ The projection of R over A' is the relation $R'(A_i:D_i,\ldots,A_j:D_j)$ where $(a_i,\ldots,a_j)$ R' iff a tuple appears in R having a_k as its A_k value for all A_k in A'.

As another example suppose a listing was needed which showed all cities in which employees live.

LOC [CITY]:= CURRENT_EMPLOYEES [CITY]

LOC

CITY
Richmond
Goochland
Ashland

LOC RELATION

Figure 15

Since the result of this operation is a relation or set and a set contains no duplicates, each city is listed only once, even if several employees reside there.

Union

Because relations are simply sets several set operators can be applied and are very useful. Suppose for instance a listing was desired of all employees who work in the Accounting Department or the Development department. Using SELECT on the relation LOCATION (Figure 7) two relations can be developed with show the employees in each department respectively. UNION then will paste these two tables together yielding the desired result. This can be denoted by

```
TEMP1 [ NAME, DEPARTMENT, DEPT# ]:=
        LOCATION WHERE DEPARTMENT = 'Accounting'

TEMP2 [ NAME, DEPARTMENT, DEPT# ]:=
        LOCATION WHERE DEPARTMENT = 'Development'

BOTH [ NAME, DEPARTMENT, DEPT# ]:=
                                   TEMP1 UNION TEMP2
```

The resulting relation is given in Figure 16.

BOTH

NAME	DEPARTMENT	DEPT#
William Smith	Development	472
Robert Brown	Accounting	410
Mary Brown	Accounting	410
John Jones	Development	472

BOTH RELATION

Figure 16

The union operation is performed on two relations that had the same attributes. The relations involved are said to be union compatable.

DEFINITION: Two relations R and S are union compatible iff both are of degree n and the ith attribute values of R are drawn from the same domain as the ith attribute values of S, for all i, $1 \leq i \leq n$.

DEFINITION: Let $R(A_1:D_1,\ldots,A_n:D_n)$ and $S(B_1:D_1,\ldots,B_n:D_n)$ be union compatible. The UNION of R and S is the relation $T(C_1:D_1,\ldots C_n:D_n)$ where $(c_1,\ldots,c_n)$ appears in T iff it appears in R or in S or in both.

Intersection

DEFINITION: Let $R(A_1:D_1,\ldots,A_n:D_n)$ and $S(B_1:D_1,\ldots,B_n:D_n)$ be union compatible. The intersection of R and S is the relation $T(C_1:D_1,\ldots,C_n:D_n)$ where $(c_1,\ldots,c_n)$ appears in T iff it appears in both R and in S.

As an example of INTERSECT suppose a relation was desired which showed all employees who had sons under the age of 10. SELECT applied to the relation FAMILY (Figure 10) in two ways can be used to find those employees that have sons and those who have dependents under the age of 10. The final result shown in Figure 17 then is the intersection of these two sets of tuples. This can be denoted by

```
SONS [ NAME, DEPENDENT, RELATION, AGE ]:=
                    FAMILY WHERE RELATION = 'son'

YOUNG [ NAME, DEPENDENT, RELATION, AGE ]:=
                    FAMILY WHERE AGE > 10

YOUNG_SON [ NAME, DEPENDENT, RELATION, AGE ]:=
                    SONS INTERSECT YOUNG
```

YOUNG_SON

NAME	DEPENDENT	RELATION	AGE
John Jones	James	son	5
Mary Brown	James	son	5

YOUNG_SON RELATION

Figure 17

Minus

Suppose now a table was needed which showed all employees that work in neither Accounting nor Development. Using the relation BOTH (Figure 16) the result can be obtained by

NOT [NAME, DEPARTMENT, DEPT#]:=
LOCATION MINUS BOTH

Minus then takes those tuples in LOCATION and essentially "cuts out" the ones that appear in BOTH.

DEFINITION: Let $R(A_1:D_1,\ldots,A_n:D_n)$ and $S(B_1:D_1,\ldots,B_n:D_n)$ be union compatable. Then R MINUS S is the relation $T(C_1:D_1,\ldots,C_n:D_n)$ where $(c_1,\ldots,c_n)$ appears in T iff it appears in R but does not appear in S.

Cross Product

It is sometimes helpful to consider the familiar cross product of the elements of two relations.

DEFINITION: Let $R(A_1:D_1,\ldots,A_m:D_m)$ and $S(B_1:E_1,\ldots,B_n:E_n)$ be two relations. R CROSS S then is the set of all tuples $(a_1,\ldots,a_m,b_1,\ldots,b_n)$ such that $(a_1,\ldots,a_m)$ appears in R and $(b_1,\ldots,b_n)$ appears in S.

Suppose, for instance, that the personnel department had several applications for programming positions available in various departments in the company. Two relations showing this information are listed in Figure 18. The personnel office, wanting to

APPLICANTS

[NAME, POSITION]

NAME	POSITION
Mary Parks	Programmer
Susan Powell	Secretary
Louis East	Programmer
Thomas Myer	Engineer
Edward Allen	Programmer

AVAILABLE

[DEPT, OPEN]

DEPT	OPEN
Accounting	Lawyer
Engineering	Engineer
Personnel	Secretary
Development	Programmer
Accounting	Secretary
Resources	Programmer

APPLICANT AND AVAILABLE RELATIONS

Figure 18

insure that each department that has openings has an opportunity to view every applicants' credentials, could construct a table showing every combination of applicant and department. This can be accomplished by selecting first those applicants for programming jobs, then selecting those departments wanting programmers and then finally taking the cross product of these two resulting

relations. This can be denoted by

```
PROG [ NAME, POSITION ]:=
          APPLICANTS WHERE POSITION = 'Programmer'

AV_PROG [ DEPT, OPEN ]:=
          AVAILABLE WHERE POSITION = 'Programmer'

ALL [ NAME, POSITION, DEPT, OPEN ]:=
          PROG CROSS AV_PROG
```

and the result is shown in Figure 19.

ALL

NAME	POSITION	DEPT	OPEN
Mary Parks	Programmer	Development	Programmer
Mary Parks	Programmer	Resources	Programmer
Louis East	Programmer	Development	Programmer
Louis East	Programmer	Resources	Programmer
Edward Allen	Programmer	Development	Programmer
Edward Allen	Programmer	Resources	Programmer

ALL RELATION

Figure 19

Note that there is redundant information listed in the relation ALL. If a projection was taken on ALL a duplicate column could be omitted.

Join

In order to "paste" two relations together over common attribute values the operator JOIN can be used.

DEFINITION: Let $R(A_1:D_1,\ldots,A_m:D_m)$ and $S(B_1:E_1,\ldots,B_n:E_n)$ be two relations such that attributes $(C_i:D_i,\ldots,C_j:D_j)$ are common to both. Let op denote any one of the comparison operators =, <, <=, >, >=, <> and let $T(A_1:D_1,\ldots,A_m:D_m,B_1:E_1,\ldots,B_n:D_n)$ be the result of joining R and S. Then the tuple

$(a_1,\ldots,c_{ri},\ldots,c_{rj},\ldots,a_m,$
$b_1,\ldots,c_{si},\ldots,c_{sj},\ldots,b_n)$ appears

in T iff c_{rk} op c_{sk} evaluates to true for all k, i k j.

The most common comparison operator used in a join is the equality operator. This operation is designated

as the equijoin. If an equijoin is used two or more columns of the resulting relation will be exactly the same. Since having this redundant information is often undesirable the natural join is defined to be the equijoin with column duplication eliminated. The use of the operator JOIN will designate the natural join throughout this text.

As an example suppose a relation was needed which listed employee names, their department, department numbers and supervisors. The needed information is listed in two separate relations LOCATION (Figure 7) and

SUPERVISORS

[DEPT#, MANAGER]

DEPT#	MANAGER
423	Adam Parker
472	Lorretta Evans
415	Mary Brown
402	Andrew Burne
450	John Adams

SUPERVISORS RELATION

Figure 20

SUPERVISORS (Figure 20) which share a common attribute, DEPT#. The desired table could be constructed by joining the two tables over their common attribute DEPT# and can be denoted by

INFO [NAME, DEPARTMENT, DEPT#, MANAGER]:=
LOCATION JOIN SUPERVISORS

The result is shown in Figure 21.

INFO

NAME	DEPARTMENT	DEPT#	MANAGER
John Jones	Engineering	423	Adam Parker
William Smith	Development	472	Lorretta Evans
Mary Brown	Resources	415	Mary Brown
Ann Williams	Personnel	402	Andrew Burne
John Jones	Development	472	Lorretta Evans
Paul Jones	Engineering	423	Adam Parker
Howard Evans	Personnel	402	Andrew Burne
William Smith	Resources	415	Mary Brown

INFO RELATION

Figure 21

Note that for some reason department number 410 is not listed in the relation SUPERVISORS. Hence there is no match for tuples in the relation LOCATION with DEPT# attribute value 410. These tuples are therefore not included in the result.

Suppose now a user wanted to elaborate on the relation INFO in Figure 21 and include the date an employee was assigned to that department. The assignment dates are listed in ASSIGNMENT (Figure 22). A join can then be performed over these relations which

ASSIGNMENTS

[NAME, DEPT#, DATE]

NAME	DEPT#	DATE
John Jones	423	5/ 5/82
William Smith	472	12/13/83
Mary Brown	415	6/15/84
Ann Williams	402	7/20/82
Robert Brown	410	8/12/83
Mary Brown	410	7/16/81
John Jones	472	2/17/82
Paul Jones	423	10/30/83
Robert Field	411	3/ 2/81
Howard Evans	402	11/14/82
William Smith	415	12/ 1/81

ASSIGNMENTS RELATION

Figure 22

share two attributes in common, NAME and DEPT#. This can be denoted by

MORE [NAME, DEPARTMENT, DEPT#, MANAGER, DATE] :=
INFO JOIN ASSIGNMENTS

and the result is shown in Figure 23.

MORE

NAME	DEPARTMENT	DEPT#	MANAGER	DATE
John Jones	Engineering	423	Adam Parker	5/ 4/82
William Smith	Development	472	Loretta Evans	12/13/83
Mary Brown	Resources	415	Mary Brown	6/15/84
Ann Williams	Personnel	402	Andrew Burne	7/20/82
John Jones	Development	472	Loretta Evans	2/17/82
Paul Jones	Engineering	423	Adam Parker	10/30/83
Howard Evans	Personnel	402	Andrew Burne	11/14/82
William Smith	Resources	415	Mary Brown	12/ 1/81

MORE RELATION

Figure 23

Since John Jones, for instance, works for more than one department the values under both NAME and DEPT# must be matched before a tuple is constructed.

The Primitive Operators

Consider now the relations in Figure 24 which contain names and department numbers of employees who serve on various committees in the company. Suppose a relation was desired which showed names of the employees

RESEARCH

[NAME, DEPT#]

NAME	DEPT#
John Jones	423
Paul Jones	423
Robert Brown	410
Ann Williams	402
Howard Evans	402
Andrew Burne	402

EM_RELATIONS

[NAME, DEPT#]

NAME	DEPT#
Robert Brown	410
Robert Field	411
Ann Williams	402
William Smith	472
Paul Jones	423

RESEARCH AND EM_RELATIONS RELATIONS

Figure 24

who serve on both committees. This can be obtained quite simply by intersecting the two tables in Figure 24 and can be denoted by

```
BOTH_COMM [ NAME, DEPT# ] :=
        RESEARCH INTERSECT EMP_RELATIONS
```

The result is listed in Figure 25.

BOTH_COMM

NAME	DEPT#
Paul Jones	423
Robert Brown	410
Ann Williams	402

BOTH_COMM RELATION

Figure 25

This result can be obtained in another way. Consider the relation obtained by performing the following operations.

```
TEMP1 [ NAME1, DEPT#1, NAME2, DEPT#2 ] :=
                     RESEARCH CROSS EM_RELATIONS

TEMP2 [ NAME1, DEPT#1, NAME2, DEPT#2 ] :=
                     TEMP1 WHERE NAME1 = NAME2

BOTH_COMM [ NAME, DEPT#1] :=
                     TEMP2 [ NAME, DEPT#1 ]
```

Clearly the above three steps result in a relation equivalent to the one resulting from the intersection operation. Therefore intersection is not considered a primitive relational operator and an alternative definition may be given.

DEFINITION: Let R and S be union compatiable relations. Then T := R INTERSECT S can be derived using T_0 := R CROSS S

T_1 := T_0 WHERE $A_1 = B_1$

T_2 := T_1 WHERE $A_2 = B_2$

...

T := T_{n-1} WHERE $A_n = B_n$.

A similar argument can be made concerning the join operations. Consider the relation INFO (Figure 21) which was derived as the result of a join. If the following operations were performed an equivalent relation would be derived.

```
TEMP1 [ NAME, DEPARTMENT, DEPT#1, DEPT#2, MANAGER ] :=
            LOCATION CROSS SUPERVISORS

TEMP2 [ NAME, DEPARTMENT, DEPT#, MANAGER ] :=
            TEMP1 WHERE DEPT#1 = DEPT#2

INFO [ NAME, DEPARTMENT, DEPT#, MANAGER ] :=
            TEMP2 [ NAME, DEPARTMENT, DEPT#1, MANAGER]
```

Consequently an alternative definition of the equijoin may be given.

DEFINITION: Let R and S be two relations and let T be the result of doing an equijoin on these relations which have attributes($C_i:D_i,\ldots,C_j:D_j$) in common. Let C_{rk} denote a common attribute in R and C_{sk} denote a common attribute in S. Then T can be derived using

$$T_0 := \text{R CROSS S}$$

$$T_1 := T_0 \text{ WHERE } C_{ri} = C_{si}$$

$$T_2 := T_1 \text{ WHERE } C_{ri+1} = C_{si+1}$$

$$\ldots$$

$$T := T_{j-1} \text{ WHERE } C_{rj} = C_{sj}$$

Doing a projection on the relation T given in the definition above results in a relation equivalent to the natural join. Substituting any other comparison operator will also result in equivalent joins.

It is clear then any relation derived using an intersection or join can be derived using alternative relational operations[61]. The other operations then are considered primitive.

DEFINITION: The primitive relational operators are PROJECT, SELECT, UNION and MINUS.

When designing a database it is necessary to balance the complexity of the system against the degree of "user friendliness" the system possesses. Consequently while the operators JOIN and INTERSECT may be derived through other means it might be worthwhile to incorporate them in the system in order to provide the user with operations that are easy to understand and use. This is especially true in the case of the natural join, probably the most commonly used nonprimitive operator.

Applying the Relational Operators

Some of the examples in previous sections have shown that deriving a desired relation might require applying operators to intermediate relations. The following examples illustrate how applying operators in a series of steps allows the user to extract almost any amount of information from the database.

Example 1: Find the names of all employees living in Richmond who work for Mary Brown.

Solution: Use the relations shown in Figures 6, 7, 20

```
TEMP1 [ NAME, STREET, CITY, ST, TELEPHONE ] :=
    CURRENT_EMPLOYEES WHERE CITY = 'RICHMOND'

TEMP2 [ NAME ] := TEMP1 [ NAME ]

TEMP3 [ NAME, DEPARTMENT, DEPT# ] :=
        TEMP2 JOIN LOCATION

TEMP4 [ NAME, DEPARTMENT, DEPT#, MANAGER ] :=
        TEMP3 JOIN SUPERVISORS

TEMP5 [ NAME, DEPARTMENT, DEPT#, MANAGER ] :=
        TEMP4 WHERE MANAGER = 'Mary Brown'

ANSWER1 [ NAME ] := TEMP5 [ NAME ]
```

ANSWER1

NAME
William Smith

It could be argued that it would be worthwhile to allow the system to nest the operators and write the derivation in one line. For example the above series of operations could be written as:

```
ANSWER1 [ NAME ] :=
       (((((CURRENT_EMPLOYEES WHERE CITY = 'Richmond')
            [ NAME ] )JOIN LOCATION) JOIN SUPERVISORS)
              WHERE MANAGER = 'Mary Brown')[ NAME ]
```

Looking at the inner most sets of parentheses and moving out it can be verified that the above is equivalent to the original six step outlined. Of course, this type of nesting would result in a much more complex system both to implement and perhaps to teach to a user. If the system was to be accessed by people experienced with writing mathematical expressions the above would probably be preferred. If, however, the typical user was not familiar with this type of construct the step by step application might be more appropriate.

Example 2: List the names and telephone numbers of all employees who have dependent children.

Solution: Use relations in Figures 6,10

```
TEMP1 [ NAME, DEPENDENT, RELATION, AGE ] :=
                  FAMILY WHERE RELATION = 'son'

TEMP2 [ NAME, DEPENDENT, RELATION, AGE ] :=
                  FAMILY WHERE RELATION = 'daughter'

TEMP3 [ NAME, DEPENDENT, RELATION, AGE ] :=
                  TEMP1 UNION TEMP2

TEMP4 [ NAME ] := TEMP3 [NAME]

TEMP5 [ NAME, STREET, CITY, ST, TELEPHONE ] :=
                  CURRENT_EMPLOYEES JOIN TEMP4

ANSWER2 [ NAME, TELEPHONE ] :=
                  TEMP5 [ NAME, TELEPHONE ]
```

Alternatively, using nested operators

```
ANSWER2 [ NAME, TELEPHONE ] :=
      ((((FAMILY WHERE RELATION = 'son') UNION
         (FAMILY WHERE RELATION = 'daughter'))
         [NAME]) JOIN CURRENT_EMPLOYEES)
         [ NAME,TELEPHONE]
```

ANSWER2

NAME	TELEPHONE
John Jones	355-1234
William Smith	257-1234
Mary Brown	223-1234
Howard Evans	987-8989

Example 3: Find the names of all employees who work for more than one department.

Solution: Use the relations in Figure 7.

```
LOC1 [ NAME1, DEPT#1 ] :=
                LOCATION [ NAME, DEPT# ]

LOC2 [ NAME2, DEPT#2 ] :=
                LOCATION [ NAME, DEPT# ]

TEMP1 [ NAME1, DEPT#1, NAME2, DEPT#2 ] :=
                LOC1 CROSS LOC2

TEMP2 [ NAME1, DEPT#1, NAME2, DEPT#2 ] :=
                TEMP1 WHERE NAME1 = NAME2

TEMP3 [ NAME1, DEPT#1, NAME2, DEPT#2 ] :=
                TEMP2 WHERE DEPT#1 <> DEPT#2

ANSWER3 [ NAME ] := TEMP3 [ NAME ]
```

ANSWER3

NAME
John Jones
William Smith
Mary Brown

In Example 3 the first two steps are necessary in order to rename the attributes differently so the result of crossing the two equivalent relations will have four distinct attribute names. Even in a system which supports nesting of operations then, these steps must be separate.

It is clear then that even in more complicated derivations the user can continue to view the process as simply "pasting and cutting" tables in order to construct new tables. Once the user has become familiar with the syntax of each operation there should be little or no problem with the understanding of how they work.

Dangers in Applying Relational Operators

All of the examples given previously in this chapter have resulted in relations which gave accurate information. In some cases however, an indiscriminate application of relational operators could lead to invalid conclusions.

Suppose a relation was given which described government contracts the company has. Listed would be the project name, the employee in charge of the

CONTRACTS

[PROJECT, MANAGER, DEPT#]

PROJECT	MANAGER	DEPT#
Radar Design	John Jones	423
Error Control	William Smith	415
Cost Effectiveness	Mary Brown	410
System Design	Paul Jones	423

CONTRACTS RELATION

Figure 26

project and the number of the department responsible for that project (Figure 26). Consider now the following projection taken over this relation.

CONTRACT2 [NAME, DEPT#] :=
CONTRACTS [MANAGER, DEPT#]

CONTRACT2

NAME	DEPT#
John Jones	423
William Smith	415
Mary Brown	410
Paul Jones	423

CONTRACT2 RELATION

Figure 27

The resulting relation (Figure 27) implies that John Jones works for department 423. By examining the relation LOCATION (Figure 7) however, it is clear that this employee is working for department 472 as well. The relation CONTRACT2 then is incomplete and consequently invalid.

In a similar way incorrect results can be derived through the use of JOIN. Suppose a relation describing where departmental offices are located in the company

OFFICE

[DEPT#, WING]

DEPT#	WING
423	East
415	North
410	South
472	North
402	West
411	West

OFFICE RELATION

Figure 28

complex (Figure 28) was joined with the relation LOCATION.

OFFICE2 [NAME, DEPARTMENT, DEPT#] :=

LOCATION JOIN OFFICE

OFFICE2

NAME	DEPARTMENT	DEPT#	WING
John Jones	Engineering	423	East
William Smith	Development	472	North
Mary Brown	Resources	415	North
Ann Williams	Personnel	402	West
Robert Brown	Accounting	410	South
Mary Brown	Accounting	410	South
John Jones	Development	472	North
Paul Jones	Engineering	423	East
Robert Field	Maintenance	411	West
Howard Evans	Personnel	402	West
William Smith	Resources	415	North

OFFICE2 RELATION

Figure 29

Note that OFFICE2 (Figure 29) implies that John Jones has an office in both the East and North wings of the complex when in reality he might have an office in only one. Again an inaccurate, invalid relation has been derived.

In both cases the questionable relations are the result of the associations among attributes in the relations CONTRACTS and LOCATION. In each case an

employee can be associated with more than one department. This can be denoted by NAME ->-> DEPT# where the double arrow implies a one-to-many association. When relations are designed which contain this type of dependency care should be taken by the users to avoid possible invalid results[62]. Of course if the user carefully labels any resulting relations ambiguities could be eliminated. For instance in the relation in Figure 29 if the WING attribute was renamed to reflect more clearly that it designates the location of the department and not necessarily the employee the relation would no longer be considered inaccurate. It is imperative then that the users clearly understand the relations and the relationships that exist among the data items in the relations in order to avoid invalid results.

NULL AND DEFAULT VALUES

Suppose a user in a personnel department is entering a new tuple in a relation describing employees in a company. The user will be prompted for the employee's social security number, name, telephone number and address. It would not be uncommon however, that the personnel department would not have all the information needed at the time it would be convenient to include an employee in the data base. The employee's telephone, for instance, might not be installed if the employee is new to the area. The question then is how the relation can represent these unknown values.

In general a null value for an attribute is a value which is unknown at the time it is observed. In the physical setting the null value can be represented by a bit pattern distinguishable from any value in the attribute's domain. The use of null values supports allowing a user to enter a tuple when attribute values are not known. It also allows the addition of a new attribute to a relation without supplying all the attribute values beforehand[63]. Note the null value is not a member of the attribute's domain but merely denotes value unknown. Recall, however, that the use of nulls is restricted to attributes which are not part of the primary key of any relation in the system.

The use of null values introduces some interesting complications to the data base system. Consider for instance the truth value of x = y if x or y or both are null. Clearly the answer is unknown or null. In order to support nulls then the meaning of each relational operator must be extended in order to handle null values in a predefined way[64].

Consider an example where it might be appropriate to include nulls. Suppose two relations in a hospital data base were available which showed patient name and operation in OP and surgeon name and the operation that surgeon performs in SURG (this example assumes that a surgeon performs only one operation).

OP

[NAME, OPERATION]

NAME	OPERATION
John Jones	Tonsils
Robert Smith	CVG
Ann Dunhill	?
Robert Dean	XRAY

SURG

[SURG_NAME, OPERATION]

SURG_NAME	OPERATION
Richard Keenan	Tonsils
Frank Kane	CVG
Betty Tyler	?

OP AND SURG RELATIONS

Figure 30

Note that the operation description was not known when

the tuple for Ann Dunhill was entered and, hence, it is null. Suppose the user wants a relation showing those patients with incomplete information. The SELECT operator can not be used since the comparison involved must compare an attribute name and a value from the attribute's domain. Since null is not an element of any domain the user in fact could not select those tuples where the comparison involves a null. Consequently the SELECT operator must be extended to consider the null values. MAYBE_SELECT chooses those tuples where nulls appear in the given attribute. This can be denoted by

MAYBE_OP [NAME, OPERATION] :=
OP WHERE_MAYBE [OPERATION]

and the result is shown in Figure 31.

MAYBE_OP

NAME	OPERATION
Ann Dunhill	?

MAYBE_OP RELATION

Figure 31

Suppose now a user wants a relation which showed patient name, operation and surgeon. A natural join of OP and SURG would result in the relation shown in Figure 32.

PAT_SURG

NAME	OPERATION	SURG_NAME
John Jones	Tonsils	Richard Keenan
Robert Smith	CVG	Frank Kane

PAT_SURG RELATION

Figure 32

Note that patient Ann Dunhill does not appear in PAT_SURG since the truth value of comparing a null with any other value is unknown and not true. A similiar argument can be made concerning the surgeon Betty Tyler. The user however might want that information included in the result, perhaps reminding the user to update the relations OP and SURG. The OUTER_JOIN accomplishes this by including in the resulting relation not only those tuples that result in a match but those that do not, extended with null attribute values. This can be denoted by

```
PAT_SURG2 [ NAME, OP, SURG_NAME ] :=
                                   OP OUTER_JOIN SURG
```

The result is listed in Figure 33.

PAT_SURG2

NAME	OP	SURG_NAME
John Jones	Tonsils	Richard Keenan
Robert Smith	CVG	Frank Kane
Ann Dunhill	?	?
?	?	Betty Tyler
Robert Dean	XRAY	?

PAT_SURG2 RELATION

Figure 33

Similiar extensions can be made to all of the relational operators. It is clear that nulls serve an important purpose in a relational data base but their use results in adding some possibly confusing complications into the system. Consider again SELECT. In order to get all tuples with an attribute value not equal to a given value the user must SELECT using <> and then UNION that relation with the result of a MAYBE_SELECT. The user must also be thoroughly aware of the correct interpretation of null. Suppose in the example of OUTER_JOIN given earlier that the fact that the operation XRAY does not appear in SURG implies merely that this procedure does not require a surgeon. The result of the OUTER_JOIN however implies that those null surgeon names are the result of a missing, unknown value. The user must be aware of this dual

interpretation and take care to respond to the results accordingly.

The use of nulls also introduces some implementation problems. A null value can be represented by a value that is known not to belong to that attribute's domain. For instance if the attribute is salary -1 could be used. If, however, it is not safe to choose a value in that way a hidden field could be used. This hidden field, one associated with each attribute in each relation, could be set to 1 if the associated field value is null. This, of course, would require extra storage, additional seeks and extra programming. In either case, the application programmer must be aware of each possible representaion of null values and program accordingly. Consider, for instance, the following lines of code which intuitively suggest that SUB_PROG will always be executed.

```
If Y = Y Then
Execute SUB_PROG
```

But will the application programmer want SUB_PROG to be executed if Y is null? Recall that the truth value of Y = Y is unknown if Y is null. Hence, the programmer must take that into account. Suppose now the appliation programmer was sorting the relation by a particular field that could be null. The programmer must take appropriate precautions to insure that tuples with nulls

are sorted in a way that makes sense.

Many of the complications discussed above are the result of the fact that the null value is not a member of an attribute's domain. This subtle distinction and its ramifications must be made clear to the users in order to insure the full, correct use of the data base system.

An alternate method of representing unknown values in a relational data base is to use default values. In this implementation each attribute that may accept unknown values has a default value in its domain that indicates the value is unknown[65]. For instance, if a patient's operation was not known at entry time this could be represented by a domain value of "unknown". This default value must be clearly defined as the default. If a user tries to insert a tuple with a missing attribute that does not have a default it must be rejected.

Several of the complications introduced by the use of nulls now no longer pose a problem. In general the truth value x = y is now either true or false. The MAYBE_SELECT operator is therefore no longer needed. Because there is no longer any ambiguity involved in saying x = y application programs will be more straightforward. Consideration must still be made for default values but in an obvious way.

Note the OUTER_JOIN is still applicable but now default values will be generated for the appropriate unknown attribute values. The user must still be aware of the differences in meaning when seeing these default values in the result of an OUTER_JOIN.

Since the default values belong to the attribute's domain they behave in more predictable ways. It is therefore the more satisfactory approach to representing unknown values in a relational data base[66].

REFERENCES

[60] Date, C. J., An Introduction to Database Systems, 3d ed., (Reading, Mass.: Addison-Wesley, 1982) p. 60.

[61] Ibid., p. 215.

[62] Martin, James, Computer Data Base Organization, 2d ed., (Englewood Cliffs, N.J.: Prentice-Hall, 1977) p. 221.

[63] Date, C. J., An Introduction to Database Systems, Volume II, (Reading, Mass.: Addison-Wesley, 1983) p. 210.

[64] Ibid. p. 221.

[65] Ibid. p. 225.

[66] Ibid. p. 226.

CHAPTER 6

NORMAL FORMS

When designing a logical data base description it would be helpful to have some way to insure that the design is consistent with the goal of maintaining accurate data. Normal form theory provides a method by which the designer of a data base can check the design[67]. This theory formalizes what intuitively appears to be a sound basis for design.

To illustrate the benefits of having a relational design satisfy various normal forms the design of an employment agency's data base will be considered. Information included will be the personal data concerning an employment candidate (name, social security number, address, etc.), the candidate's skills and background, information concerning the jobs that are open as well as data about the employment agents themselves.

FIRST NORMAL FORM

It has already been noted in the definition of a relation (Chapter 5) that any attribute value listed in a tuple of a relation must be a single valued member of the domain of the given attribute.

DEFINITION: A relation is in First Normal Form iff it contains atomic values only.

Consider then a relation which will contain skills that each candidate has. Certainly one candidate might have several skills. A reasonable relation then would be SKILLS shown in Figure 34.

SKILLS

[SOC.SEC.#, SKILL]

SOC.SEC.#	SKILL
123456789	Typing
234567890	Steno
234567890	Typing
234567890	Filing

SKILLS RELATION

Figure 34

Note that in this relation the entire tuple is the key and that to list all the skills for one candidate that candidate's social security number might appear several times.

FUNCTIONAL DEPENDENCIES

Before proceeding it is necessary to define any dependencies that exist among the attributes of a tuple.

Consider a relation that includes information concerning job listings the agency now has:

CO_INFO
[<u>COMPANY NAME</u>, <u>POSITION</u>, COMPANY_LOC, SALARY, STATUS]

Suppose that STATUS is determined independently by both the salary and the position in the company, i.e. a system analyst at company X making $30,000 is classified as professional both because the position at the company is classified as such and because of the salary range.

Intuitively it is clear that COMPANY_LOC depends on COMPANY_NAME. More rigorously, COMPANY_LOC is functionally dependent on COMPANY_NAME.

DEFINITION: Given a relation R and sets of attributes A and B, B of R is functionally dependent on A of R iff each A value in R has associated with it at most one B value in R at any one time.

For example suppose Company X is listed as being located in New York City. Then since COMPANY_LOC is functionally dependent on COMPANY_NAME each time Company X appears under COMPANY_NAME, New York City will appear under COMPANY_LOC. This is denoted by COMPANY_NAME -> COMPANY_LOC.

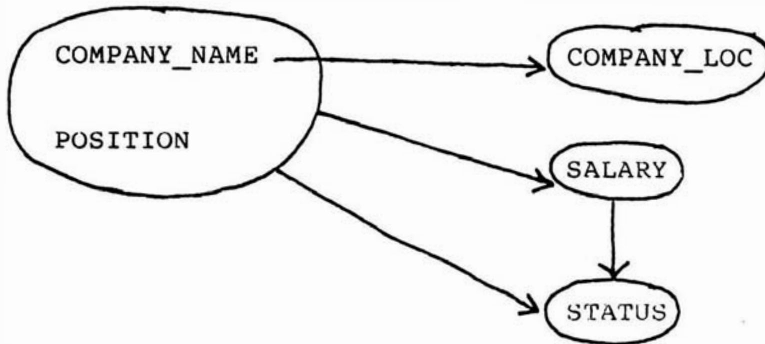
Now since COMPANY_NAME and POSITION form the key of the relation it is indeed true that COMPANY_LOC is functionally dependent on the key as well as on COMPANY_NAME. COMPANY_LOC, however is said to be fully functionally dependent on COMPANY_NAME and COMPANY_NAME is referred to as a determinant.

DEFINITION: B is fully functionally dependent on A iff it is functionally dependent on A and not functionally dependent on any proper subset of A.

DEFINITION: A is a determinant iff there exists some other attribute B such that B is fully functionally dependent on A.

Henceforth the term functionally dependent will imply fully functionally dependent and will be denoted by A -> B.

There are several other functional dependencies in the relation CO_INFO. These can be summed up in a functional dependency diagram shown in Figure 35.



CO_INFO FUNCTIONAL DEPENDENCY DIAGRAM

Figure 35

The large bubble denotes a set of attributes. The fact that COMPANY_LOC is functionally dependent on a subset of that set is denoted by the fact that the arrow starts at the attribute COMPANY_NAME rather than from the outside of the bubble as for SALARY.

It is clear then that SALARY is functionally

dependent on the key. Also shown in the diagram is the fact that STATUS is independently functionally dependent on both (NAME, POSITION) and (SALARY).

The functional dependency diagram is a helpful tool for clearly, concisely showing the functional dependencies in a relation.

SECOND NORMAL FORM

Consider the relation CO_INFO (Figure 35). There are several problems with the design of this relation. It is not unreasonable, for instance, to suppose the employment agency would want to maintain information involving companies that might not presently have openings. As the relation is now defined it would be impossible to list that data. As soon as a position is filled that record would be deleted and if that was the only position at the company listed all information pertaining to that company would be lost. Clearly this is undesirable.

Suppose also that there are several positions open in a given company and that the company changes its location. In order to update that data in CO_INFO a search would have to be made for each occurrence of that

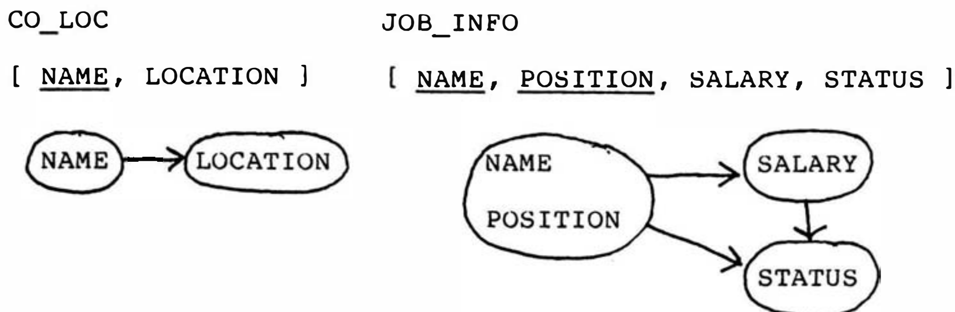
company and the tuple updated. This can jeopardize the accuracy of the data base if the update is done incompletely or incorrectly.

The potential for these problems is attributed to the same cause. As can be seen in the functional dependency diagram (Figure 35) the attribute COMPANY_LOC is functionally dependent on a subset of the key. It is this type of situation that is addressed in the definition of second normal form.

DEFINITION: A relation R is in second normal form iff it is in first normal form and every nonkey attribute is functionally dependent on the primary key.

The relation CO_INFO then violates second normal form because COMPANY_LOCATION is functionally dependent on a subset of the key.

CO_INFO can be decomposed into smaller relations without any loss of information in the following way.



DECOMPOSITION OF CO_INFO RELATION

Figure 36

It is clear that when the company's location changes only one update need be made to the data base. Information can also now be maintained about companies that do not have active job listings.

THIRD NORMAL FORM

Consider now the relation JOB_INFO (Figure 36). Each nonkey attribute is functionally dependent on the key but the attribute STATUS is also functionally dependent on the attribute SALARY. The functional dependency of STATUS on the key is said then to be transitive through the attribute SALARY. Each key value determines a SALARY and this then determines a STATUS.

This transitivity property among the attributes can be a source of problems.

Suppose, for instance, that the status levels were to be reconstructed due to, say, a cost of living adjustment. In order to do so each record would have to be searched for the status that was updated causing again the possibility for error or inconsistency.

Consider also if a company's job was filled and that record was the only record with that status level. When the record was deleted all information concerning the salary level and the status would be lost. These potential problems are the result of the transitive dependencies that exist in JOB_INFO. Third normal form addresses this issue.

DEFINITION: A relation is in third normal form iff it is in first normal form and every nonkey attribute is nontransitively dependent on the primary key.

Attributes in a relation that satisfies third normal form then are facts about the key and nothing else[68]. The attribute STATUS can be deleted from JOB_INFO and dealt with in another relation.

JOB

[NAME, POSITION, SALARY]

SAL_INFO

[SALARY, STATUS]

NAME POSITION → SALARY

SALARY → STATUS

DECOMPOSITION OF JOB_INFO

Figure 37

These two new relations provide the same information and satisfy both second and third normal forms. Now when an update needs to be made to a status level relevant records can be directly revised. Also information pertaining to levels not currently active can be maintained.

NONLOSS DECOMPOSITION OF RELATIONS

Throughout the previous sections the concept of decomposing a relation without losing any information was used. Very often it is intuitively clear how to do this but in some cases guidelines can be helpful.

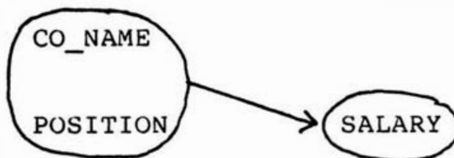
Consider again the relation JOB_INFO (Figure 36) and its functional dependency diagram. Since JOB_INFO

does not satisfy third normal form it would be advantageous to decompose this relation into two smaller relations that do satisfy third normal form. In the last section the decomposition in Figure 37 (Decomposition I) was used. An alternative decomposition is shown in Figure 38.

DECOMPOSITION II

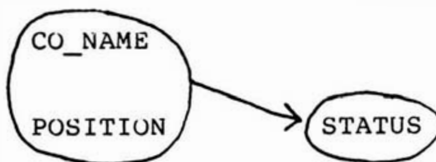
JOB_STATUS

[CO NAME, POSITION, SALARY]



JOB_STATUS

[CO NAME, POSITION, STATUS]



ALTERNATIVE DECOMPOSITION OF JOB_INFO

Figure 38

The first question which must be asked is do these two decompositions provide the same information as the original relation JOB_INFO did. A method to check this

is to do a join over common attributes and determine if the original data can be retrieved. It can be verified in both cases that this is indeed the case. Consider however a third decomposition shown in Figure 39.

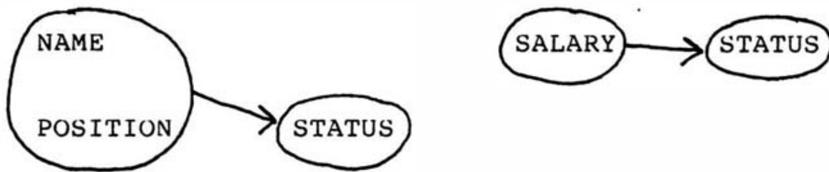
DECOMPOSITION III

JOB_STATUS

[NAME, POSITION, STATUS]

SAL_STATUS

[SALARY, STATUS]



ALTERNATIVE DECOMPOSITON OF JOB_INFO

Figure 39

It can be verified that this decomposition is not lossless because it is possible for several salaries to have the same status and hence it would not be possible to recreate the exact salary associated with a particular position at the company. Clearly this is unacceptable.

Consider again Decompositions I and II. The data base designer when choosing one over the other would hope that in addition to not losing any information

that new situations are not created that jeopardize the data's accuracy. For example, in Decomposition II if an update was made to a salary in JOB_SALARY a mechanism must be set up to insure that the functional dependency (NAME, POSITION) -> STATUS is not violated because recall status is functionally dependent on both the salary and the position in a company. In Decomposition II then the two relations are dependent, an update to one can effect the validity of the other. Consider Decomposition I however. If salary is updated for a particular position only one change need be made to the data base. The STATUS functional dependency is automatically enforced.

Decomposition I involves independent relations. This is the result of defining the relations in such a way so that the common attribute SALARY is the key for one of the relations . The original functional dependency (NAME, POSITION) -> STATUS then can be logically deduced from the dependencies generated in Decomposition I.

A guideline, then, for decomposing a relation R into R1 and R2 is to insure

i) every functional dependency in R can be deduced from R1 and R2

ii) the common attributes of R1 and R2 form a candidate key for at least one of R1 and R2

iii) it is a nonloss decomposition[69].

In this way relations can be developed that satisfy various normal forms and at the same time do not lose any information originally provided nor cause their own set of update problems.

BOYCE/CODD NORMAL FORM

Consider now a relation that would contain information pertaining to the agents of the employment agency. Certain constraints could be placed upon the assignments of the agents such as

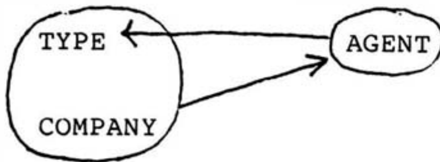
1. All jobs of a certain type in a given company are handled by the same agent.
2. Each agent handles only one type of job (i.e. will fill only programmer positions).
3. Each type of job can be handled by several agents.

Each tuple in the relation AGNTS (Figure 40) would provide the information that a specified job type is

available in a specified company and handled by a specified agent. Note that there are two candidate keys for this relation, (TYPE, COMPANY) and (COMPANY, AGENT) and the former was chosen as the primary key.

AGNTS

[TYPE, COMPANY, AGENT]



FUNCTIONAL DEPENDENCY DIAGRAM FOR AGNTS RELATION

Figure 40

While TYPE is indeed transitively dependent on the key it is itself a key attribute. Hence, the above relation satisfies third normal form. Consider however what would happen if a job listing was deleted from this relation. The information that a certain agent handles that job type could be lost. This is the result of having overlapping candidate keys with a determinant involving a subset of those keys. Boyce/Codd normal form recognizes this abnormality with

relations that are acceptable under third normal form and is a stronger defintion of third normal form[70].

DEFINITION: A relation is in Boyce/Codd normal form iff every determinant is a candidate key.

The relation AGNTS can be easily decomposed into relations that satisfy Boyce/Codd normal form (Figure 41).

AGNT

[AGENT, TYPE]

CO_AGENT

[COMPANY, AGENT]

CO_TYPE

[COMPANY, TYPE]

DECOMPOSITION OF AGNT RELATION

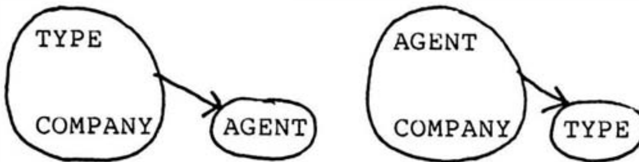
Figure 41

Suppose now that the second constraint is removed. Agents can now handle several job types and the fact that an agent is assigned a job type is no longer important. A relation similar to AGNT can be defined with candidate keys (TYPE, COMPANY) and

(AGENT, COMPANY). The functional dependency diagram for the information now defining agents and their responsibilities is shown is Figure 42. Note now

AGNT2

[COMPANY, TYPE, AGENT]



FUNCTIONAL DEPENDENCY DIAGRAM FOR RELATION AGNT2

Figure 42

that there are overlapping candidate keys but since the keys are the only determinants Boyce/Codd Normal Form is satisfied. The deletion of a job in a company would not result in the lose of any meaningful information.

FOURTH NORMAL FORM

Suppose data is to be maintained concerning certain computing skills candidates might have. A table of that information is given in Figure 43.

NAME	LANGUAGES	MACHINES
Smith	Fortran Assembler Basic	IBM
Jones	Pascal	IBM Digital

TABLE OF COMPUTER SKILLS

Figure 43

It has been assumed that languages and machines are independent of each other. In order to normalize the table in Figure 43 a cross product must be taken of Languages and Machines resulting in the relation shown in Figure 44.

SKILLS

[NAME, LANGUAGES, MACHINES]

NAME	LANGUAGES	MACHINES
Smith	Fortran	IBM
Smith	Basic	IBM
Smith	Assembler	IBM
Jones	Pascal	IBM
Jones	Pascal	Digital

SKILLS RELATION

Figure 44

The key of the relation SKILLS is the entire tuple and it is clear that the above satisfies Boyce/Codd Normal

Form.

There are certainly no functional dependencies in the relation but it is apparent that for each NAME value there is a well defined set of languages associated with it independent of the set of MACHINES also associated with it. LANGUAGES then is multidependent on NAME and is denoted by NAME ->-> LANGUAGES.

DEFINITION: Given a relation R with sets of attributes A,B,C the multivalued dependence A ->-> B holds in R iff the set of B values matching a given (A,C) pair in R depends only on the A value is independent of the C value.

Note that in order to have a multivalued dependency there must be at least 3 sets of attributes in the relation.

In addition to NAME ->-> LANGUAGES there is the multivalued dependency NAME ->-> COMPUTERS. This can be denoted as NAME ->-> LANGUAGES COMPUTERS.

There are several problems with the relation as it is now defined. If, for example, a new Language for a

particular candidate was to be included several inserts would have to be made, one for each computer. Similarly to delete a language would involve the deletion of several tuples. Clearly this is undesirable in order to maintain the data's accuracy. Fourth normal form addresses this problem.

DEFINITION: R is in Fourth Normal Form iff the only dependencies in R are functional dependencies from a candidate key to an attribute.

In other words Fourth Normal Form requires that the only determinants in the relation are the key and that no multivalued dependencies exist. The relation SKILLS can be easily broken up into relations

LANG [NAME, LANGUAGE] and COMP [NAME, COMPUTER]

where the entire tuple is the key and since there are only two attributes in each there are no multivalued dependencies.

FIFTH NORMAL FORM

Consider again a relation that describes the agent information. Suppose the constraint is that if an agent finds jobs of a certain type and deals with a company that has jobs of that type then he attempts to place candidates in jobs of that type for that company. The relation AGNT is shown in Figure 45.

AGNT

[NAME, TYPE, COMPANY]

NAME	TYPE	COMPANY
Smith	Programmer	A
Smith	Secretary	B
Jones	Secretary	A
Jones	Secretary	B

AGENT RELATION

Figure 45

The tuple (Smith, Programmer, A) then means that agent Smith is trying to fill the programmer position at Company A. The entire tuple is the key and all attributes are needed to convey the information. Furthermore it can not be said that there are

multivalued dependencies in this relation because attributes TYPE and COMPANY are not independent.

There are problems in updating this relation. Consider inserting the tuple (Jones, Programmer, B). What this is saying is that Company B has a programmer job available and Jones will try to fill it. But Smith also fills programmer jobs and handles Company B so the tuple (Smith, Programmer, B) must also be inserted. Similar situations arise upon deleting a tuple. Clearly these types of updates are undesirable because they can be easily done incompletely.

In the previous examples of normal forms it has been a nearly trivial matter to decompose the relations into smaller relations satisfying normal forms yet providing the same information. In this case however care must be taken. Consider for instance the decomposition shown in Figure 46.

ONE [NAME, TYPE]

TWO [TYPE, COMPANY]

INVALID DECOMPOSITION OF RELATION AGNT

Figure 46

These two relations will not provide all the information originally provided. Just because Smith deals with programmer jobs and there is a programmer job at Company C does not imply Smith will try to fill that position. Smith may not represent that company. If however another relation (Figure 47) was constructed a series of joins can reconstruct the original relation.

ONE

[NAME, TYPE]

TWO

[TYPE, COMPANY]

THREE

[COMPANY, NAME]

VALID DECOMPOSITION OF AGNTS

Figure 47

AGNT (Figure 45) then satisfies a join dependency since it can be reconstructed from the join of certain of its projections.

DEFINITION: R satisfies the join dependency $*(X,Y,\ldots,Z)$ iff it is the join of its projections on $X,Y,\ldots,Z$ where $X,Y,\ldots,Z$ are subsets of the attributes of R.

AGNT then satisfies the join dependency
*((NAME,TYPE),(TYPE,CO),(CO,NAME)).

Of course several relations can be decomposed this way. Consider a relation PERSON defined as PERSON [SS#, STREET, CITY, STATE]. Clearly PERSON satisfies the join dependency
*((SS#,STREET),(SS#,CITY),(SS#,STATE)) but there is no advantage in decomposing the relation in this way since there are no update or deletion problems in PERSON. The join dependency here is implied by the key and the decomposition is not necessary. Fifth Normal Form formalizes this concept.

DEFINITION: R is in Fifth Normal Form iff every join dependency in R is implied by the candidate keys of R.

Unfortunately finding the join dependencies in a relation is a nontrival task. Caution should be taken when a relation is developed that contains interrelated multivalued facts[71], i.e. one agent can deal with several types of jobs which can be placed in several different companies. When relations of this type arise the designer should carefully attempt to find any dependencies that exist and modify the relation if

necessary.

By verifying that each relation in the data base's logical design satisfies the various normal forms several potential abnormalities can be safely avoided.

REFERENCES

[67]Date, C. J., An Introduction to Database Systems, 3d ed., (Reading, Mass.: Addison-Wesley, 1982) p. 479.

[68]Martin, James, Computer Data Base Organization, 2d ed., (Englewood Cliffs, N.J.: Prentice-Hall, 1977) p. 242.

[69]Date, p. 254.

[70]Ibid., p. 249.

[71]Ibid., p. 263.

CHAPTER 7

DEVELOPMENT OF A DATA BASE

In the Department of Mathematical Sciences at Virginia Commonwealth University a manual system for scheduling classes is currently used. This method involves filling in a grid showing classes being offered with faculty assigned to teach the classes. The choice of faculty member to teach a particular class is based on several criteria. The Associate Chairman of the department, who is responsible for the schedule, must constantly keep in mind these criteria and insure they are met. It is hoped that a computerized mechanism can be developed to aid the Associate Chairman in this time consuming task.

DESCRIPTION OF THE PROBLEM

In order to design a computer system to assist in the scheduling process it was necessary to first define the manual system currently used and list the criteria

involved. The following describes the steps now followed to develop the schedule.

The Department of Mathematical Sciences contains three divisions Statistics, Computer Science and Mathematics. Each class offered through this department is designated as a class in one of these divisions. Calculus, for instance will be listed as MAT 200 indicating it is a Mathematics course at the 200 level. Operating Systems is designated as CSC 601 indicating it is an graduate level Computer Science course.

Early in the semester preceding the semester to be scheduled classes which will be offered are determined. Most offerings remain the same from year to year. For instance the Fall, 1985 schedule will mirror the Fall, 1984 schedule especially for lower level classes. The faculty generally decide any changes in upper level classes offered and inform the Associate Chairman.

Consequently, one year's schedule of class offerings is used as a basis for the following year's. Deletions, insertions and modifications are made to that schedule in order to produce the new version. A grid is set up, one for each day of the week, showing the classes offered.

Generally the faculty in each discipline determine among themselves who will probably teach the various upper level courses (300 and up). They then inform the

Associate Chairman of their decisions and the grid is filled in accordingly.

After the upper level classes have been scheduled the lower level classes are assigned. In some cases the faculty also determine the assignment for some of these classes. For instance the Computer Science members might decide among themselves who will teach certain lower level Computer Science courses knowing that each must teach one or more of them. The Associate Chairman will assign these accordingly.

Even after these assignments have been made however several classes will still need to be assigned. The decision as to who will teach these unassigned courses is based on the criteria listed below.

Criteria for Scheduling Decisions

1. Some faculty members will be better suited to teach a particular class than others. This is generally determined by the special field or discipline that the class covers. For instance, certain faculty members would be better suited to teach classes in applied mathematics than topology. Upper level classes are generally taught by faculty in the corresponding discipline. Situations where this is not the case are almost always decided ahead of time. For instance a special topics course in Computer Science could indeed

be taught by someone with a Statistics background but this would be generally be determined and scheduled well in advance.

2. Associated with each faculty member is a number which determines the maximum number of courses that faculty member could be assigned. Full time, tenure track faculty, for instance, usually teach three classes. Of course if that faculty member is on sabbatical that number could be one or even none. Other full time faculty could be assigned up to four classes and adjuncts one or two. Consequently when assigning a faculty member to a particular course care must be taken to insure that that faculty member has not already been assigned the maximum.

3. An attempt is made to insure that faculty members' schedules are not too cumbersome. For instance it could be inconvenient if it was necessary for someone to teach four nights of the week. Generally then faculty are assigned evening classes only two days a week. Also in order to give full time faculty member time to do research an attempt is made to insure each faculty member has one weekday off.

4. Some faculty will not be available to teach at specific times. For instance most adjunct faculty are

available to teach evening classes only.

A RELATIONAL DATA BASE AS A BASIS FOR SCHEDULING SYSTEM DESIGN

In order to design a system that could be used as a tool in the scheduling process several approaches could be taken. It would not be very difficult to build one or more files containing the required data and then write an application program based on these files to provide the information required in the scheduling process. These files could be updated when necessary and the application program rerun each semester.

Consider, however, some of the data needed in order to solve the scheduling problem. Clearly much of this data can be applied to other areas of concern within the Department of Mathematical Sciences. For instance suppose the Chairman of the department needed a simple listing of all faculty names and the classes they will teach the next semester. Most likely this data can be derived from one or more of the files used in the scheduling system. The Chairman would then request that a programmer, familiar with the design of the scheduling system, write the necessary program. Of course this programmer might not be readily available and consequently the Chairman would probably resort to

manually listing the needed information. This time consumming task could have been avoided if the scheduling system was designed in a way so that unanticipated requests could be easily and quickly answered by an inexperienced user.

In order to make the best use of the data needed to solve the scheduling problem a relational data base system was chosen as a basis for the system. In that way the questions posed by the scheduling process could be addressed as well as any unanticipated questions that could be answered using the same data.

The use of a relational data base system also supports future additions of data to the system as the need arises. Suppose for instance that an automated process was desired to provide information concerning textbooks used by each class otffered by the department in any given semester. Rather then build an entirely new system to support this new process the data base designeer could expand the logical and physical designs of the data base to include this information, update the DML where necessary and inform the users that this information is now available. Any existing software would continue to run without users being aware of any changes made.

The system then will not be a scheduling process but a general data base providing information about faculty and classes in the Department of Mathematical Sciences.

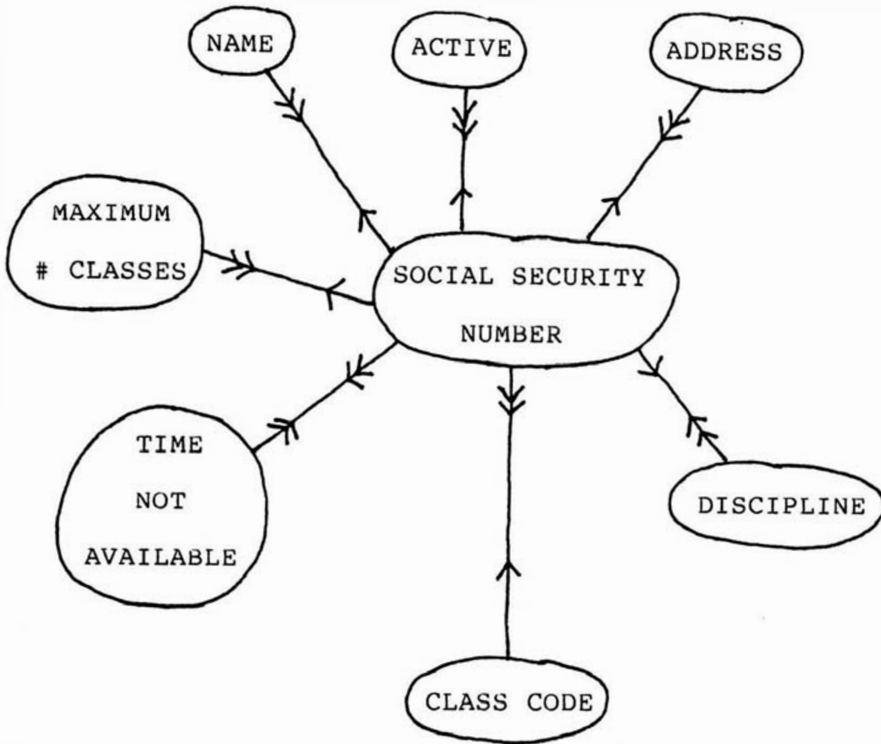
DATA NEEDED TO SOLVE THE PROBLEM

In order to set up the logical level of the relational data base it is necessary to consider what data items will be needed. It would be helpful as well to consider any additional information that the system might maintain, not necessarily for the scheduling process but for future requests. For example, the scheduling problem could very well be solved based on the unique faculty social security number but certainly it would be more helpful to include faculty names for any output procedures.

With the scheduling process in mind, consider what information is needed with respect to the faculty. Clearly the social security number can act as a unique identifier for each faculty member. As has been seen their names should also be included. The address and telephone number for each faculty member would also be helpful. Items necessary for the scheduling process include the maximum number of courses each faculty

member can teach in a semester as well as the times each faculty member is not available to teach. In order to address the issue of which faculty members are better suited to teach a given class some mechanism should be maintained whereby discipline can be determined. Also included should be an indication of what classes a given faculty member is assigned to teach. Lastly it would be advantageous to maintain information pertaining to both active and inactive faculty members. In this way the department can maintain information on faculty members who do not necessarily teach every semester. The diagram in Figure 48 shows the data items needed to represent faculty information as well as the relationships that exist among these data items. A single arrow indicates a functional dependency. For instance associated with each social security number is one faculty name. A double arrow indicates a one to many relationship. For example associated with each discipline are one or more faculty members capable of teaching a course in that discipline. Note that the relationship from NAME to SOCIAL SECURITY is one to many since it oculd be the case that two or more faculty members could have the same name. Hence that one name would be associated with many social security numbers.

Consider the relationships between TIME NOT AVAILABLE and SOCIAL SECURITY NUMBER. The double arrows



RELATIONSHIP AMONG FACULTY DATA ITEMS

Figure 48

pointing from SOCIAL SECURITY NUMBER to TIME NOT AVAILABLE imply that for each soical security number there could be many time periods when that faculty member is unavailable for teaching. In the opposite direction the double arrows pointing from TIME NOT AVAILABLE to SOCIAL SECURITY NUMBER indicate that for each time period there could be many faculty members who are not available at that time. It can be said that the relationship between TIME NOT AVAILABLE and SOCIAL SECURITY NUMBER is a many to many relationship.

With respect to the classes scheduled a unique identifier is needed to distinguish a particular scheduled class from another. An obvious choice would be the university's course code. This ten character field is defined as follows:

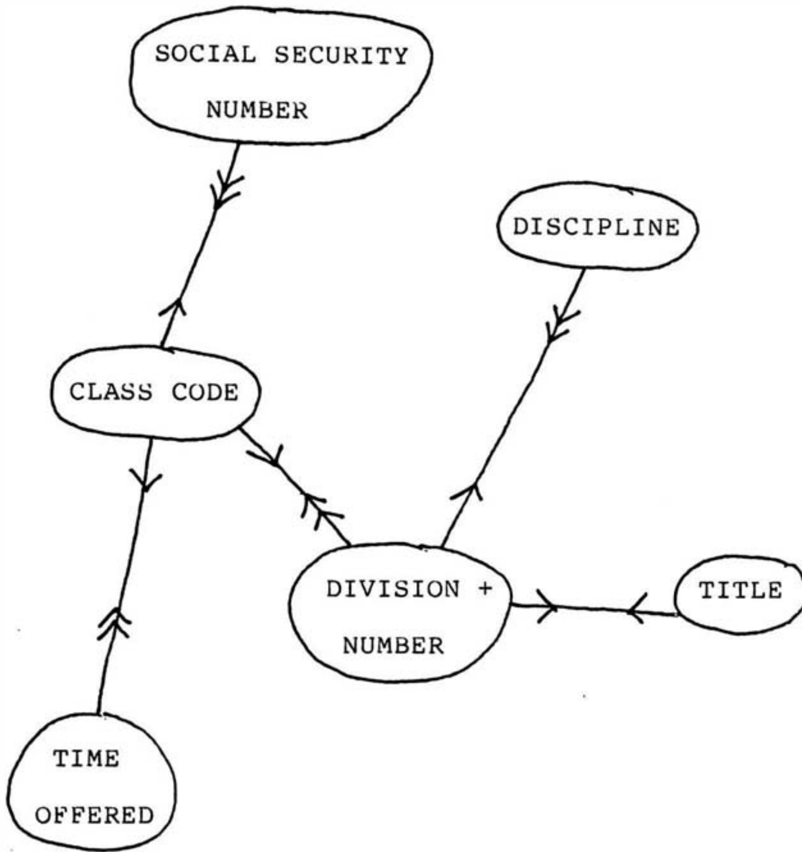
Character	
1 - 3	Division (MAT, STA, CSC)
4 - 6	3 digit course number
7 - 10	3 digit section number followed by E if evening class

Using this course code day section 5 of Computer Science 201 would be listed as 'CSC201005 '. Note that the last character, a space, denoting that the class is not an evening class is necessary since evening sections are

numbered the same as day sections. Note that this class code is really the concatentation of several pieces of information, the division designator, the course number and the section. All three are needed to uniquely identify the course. In addition to this the time of day and the days of the week the course is offered must also be maintained. In the same way that the inclusion of faculty names in the system is helpful, the course title should be listed. Lastly the dicipline or special field the course covers should be included.

The diagram in Figure 49 lists the data needed to represent class information and the relationships between those data items. Note that two bubbles actually contain the department and course number data, since that information is included in the class code. It could be argrued then that the DIVISION + NUMBER bubble is not needed. Recall however that the class code is the university's mechanism for representing only those courses being offered in a given semester. Without the DEPARTMENT + NUMBER bubble the system would have not way of maintaining information pertaining to a specific course if that course was not offered in a given semester. Consequently both bubbles should be included.

It is also true that the title of the course can be



RELATIONSHIPS AMONG CLASSES DATA ITEMS

Figure 49

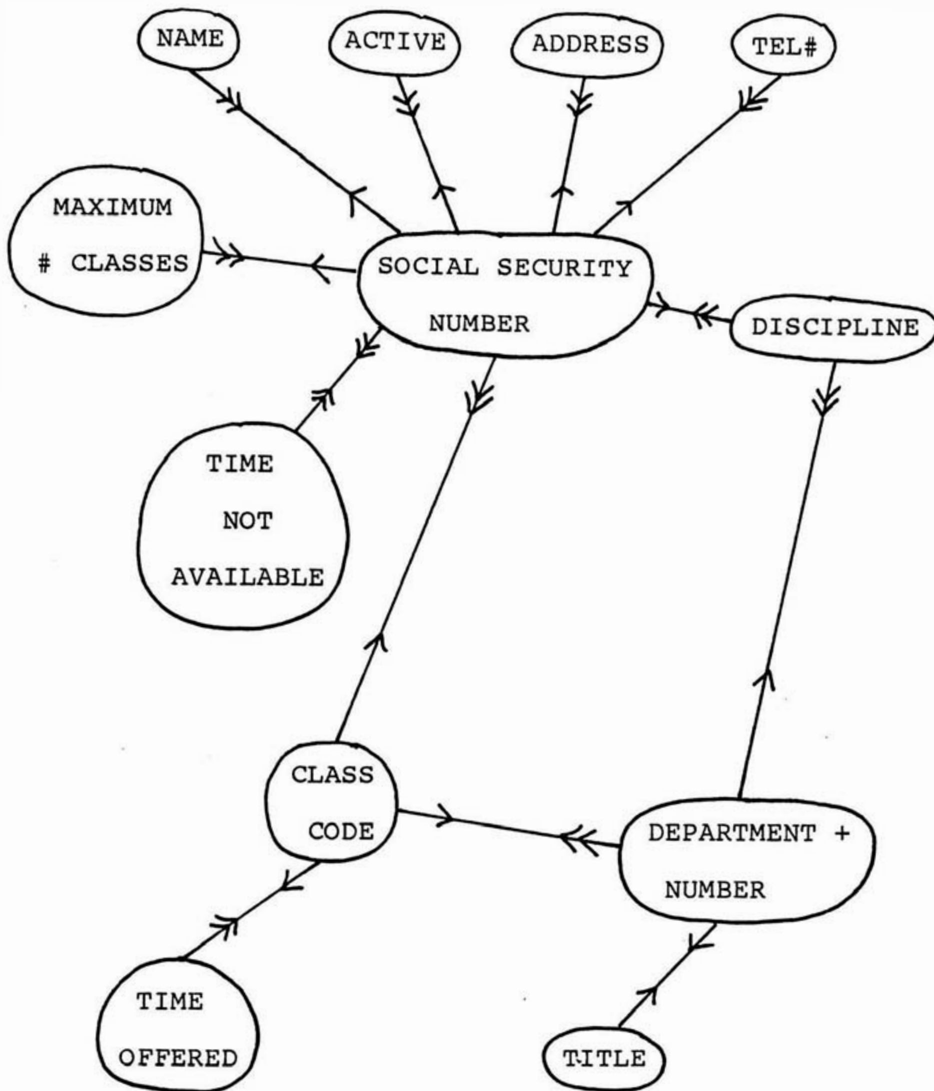
determined from the class code. Perhaps then a link should be drawn between the CLASS CODE and TITLE bubbles. While the relationship would certainly be valid it would also be redundant. Since it has been determined that the DEPARTMENT + NUMBER bubble must be included, given a class code, the department and number can be determined and hence the course's title. The same argument can be made concerning relationships between CLASS CODE and DISCIPLINE.

Figures 48 and 49 can now be joined together over their common data items to give a picture of all the data items in the system and how they are related to each other (Figure 50).

CONSTRUCTING THE LOGICAL VIEW

Using the relationships shown in Figure 50 the logical view of the data base system can be derived. Note that formal definitions have not yet been given for data items in the bubbles. Once it is known what the relations will be a precise definition of each attribute can be derived.

Consider the functional dependency between SOCIAL



RELATIONSHIPS AMONG ALL DATA ITEMS

Figure 5Ø

SECURITY NUMBER and ADDRESS. Clearly each social security number will uniquely determine one address that is associated with that social security number. The social security number then is acting as a key in a relation. In general any bubble that originates a functional dependency can serve as a key for a relation. Any bubble that orginates a one to many relationship is providing information about an entity but does not uniquely identify it. It then can serve as a nonkey attribute in a relation.

The SOCIAL SECURITY NUMBER could be the key of a relation with attributes MAX#_CLASSES, NAME, ACTIVE, ADDRESS, TEL# and DISCIPLINE. Note however that these attributes can be grouped into two categories, those dealing with personal information about the faculty member and those dealing specifically with the scheduling problem. In order then to reflect more closely the actual meaning of the data the following two relations will be used:

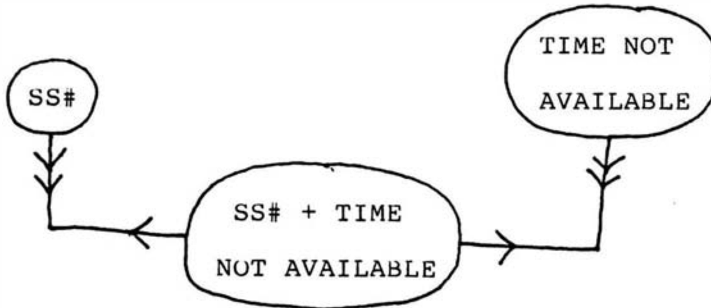
PERSONAL

[<u>SS#</u> NAME ADDRESS TEL# ACTIVE]

FACULTY

[<u>SS#</u> MAX#_CLASSES DISCIPLINE].

Consider now the many to many relationship between SOCIAL SECURITY NUMBER and TIME NOT AVAILABLE. Since neither bubble orginates functional dependency neither can serve as a key. In order to build a relation then a modification in the relationships can be made.



MODIFICATION OF SS# AND TIME NOT AVAILABLE RELATIONSHIP

Figure 51

It can be verified that the new relationship in Figure 51 provides the same information as the original. For instance given a social security number, there will be many SS# + TIME NOT AVAILABLE fields associated with it. In each of these fields however, the social security number will be the same. Consequently there are many times associated with that social security number. Now SS# + TIME NOT AVAILABLE acts as a key of a relation since this bubble orginates a functional

dependency. The following relation then can be used.

NOT_AVAIL

[SS# TIME]

In the relationships that deal with classes the following two relations can be used.

SCHEDULE

[DIV NUM SEC TIME SS#]

CLASSES

[DIV NUM TITLE DISCIPLINE]

Note rather than use class code the actual values that make up that code, DIV, NUM, and SEC are used to be consistent with their individual uses elsewhere.

Each of the relations derived can be formally defined. In some cases modifications can be made to more accurately present the data and clear up any ambiguities.

PERSONAL

Only a general understanding has been given for the ACTIVE attribute. This yes or no value will indicate whether a faculty member is currently employed by

the Department of Mathematical Sciences. A faculty member on sabbatical in a given semester, for instance, sould still be employed by the department but would teach no courses that semester. The ACTIVE value for that faculty member would be yes but MAX#_CLASSES in the relation FACULTY would be 0. In order to maintain for instance a mailing list of all former employees the tuples with an ACTIVE value of no are maintained.

With respect to the attribute ADDRESS it could be argued that it would be more flexible to list seperately the street, city, state and zip code for each faculty member. In that way to find all employees who live in Richmond say, a select can be made on the CITY attribute.

PERSONAL can be redefined then as

PERSONAL

[<u>SS#</u> NAME STREET CITY STATE ZIP TEL# ACTIVE]

<u>FACULTY</u>

In this relation the attribute discipline needs to be more clearly defined. Recall that this attribute will be used to distinguish faculty members best suited to teach a particular course. Within the Department of

Mathematical Sciences there are several special areas of expertise. These special fields include, for example, Applied Mathematics, Pure Mathematics, Operations Research and Theoretical Computing. Of course many of the courses offered by the department fall into much more general categories. An example would be Calculus which could be taught by virtually anyone in the department. In this case the special field could be the same as the division.

In order then to reflect more clearly the expertise of each faculty member the attribute can be broken up into two attributes. The first DIV will list the division the faculty member is associated with, either MAT, STA or CSC. The second attribute DISCIPLINE will list a special field such as Applied Mathematics in which the faculty member is qualified to teach.

FACULTY then can be redefined as

FACULTY
[<u>SS#</u> MAX#_CLASSES DIV DISCIPLINE]

<u>NOT AVAIL</u>

A thorough description must be made of the times when a faculty member is unavailable. Not only must a time of day be listed but the day of the week as well

since a faculty member may be unavailable at different times on different days.

With respect to the actual time of day it would be advantageous to consider the beginning and ending times that a faculty member is unavailable. In this way the user will have a more straightforward mechanism on which to query. For instance in order to find faculty members unable to teach from 8:00 to 9:00 tuples could be selected where the beginning of the unavailable time is less than or equal to 9:00 and the end is greater than 8:00.

The days of the week referred to could be representd by a character string consisting of 'MTWRF' where the days not applicable are blanks.

NOT_AVAIL

[<u>SS#</u> <u>BEG TIME</u> <u>END TIME</u> <u>DAYS</u>]

<u>SCHEDULE</u>

In this relation, as in NOT_AVAIL, the time attribute must be reconstructed. In order to be consistent the same strategy will be used.

Recall now that in the initial development of, say, the Fall semester's course schedule the previous year's

Fall schedule is used as a basis. It would be advantageous then to include an attribute in this relation indicating whether the course scheduled is for a Fall or Spring semester. In that way the next year's schedule can be built from the previous year's by merely updating the particular semester's tuples. Note however that a given section of a course may be offerd in each semester at the same time. Consequently the semeseter attribute must be a part of the key.

SCHEDULE

[<u>DIV</u> <u>NUM</u> <u>SEC</u> <u>SEMESTER</u> BEG END DAYS SS#]

CLASSES

Note that this relation contains the discipline attribute which was defined in the development of FACULTY.

CLASSES

[<u>DIV</u> <u>NUM</u> TITLE DISCIPLINE]

Integrity Rules

When formally defining the logical design of a data base not only must each attribute be defined and the

keys specified but any integrity rules concerning the relations must be clearly specified so that later they can be supported by the data manipulation language.

Consider the relations dealing with the faculty. Clearly if a faculty member is listed as active in PERSONAL a tuple should be included in FACULTY providing scheduling information about that faculty member. Conversely if a faculty member is listed in FACULTY a tuple should exist concerning that faculty member in PERSONAL.

With respect to classes scheduled, if a class's information is included in SCHEDULE clearly it should be listed in CLASSES. The converse this time is not true however since it could be the case that a course is offered so infrequently that it is not listed in the SCHEDULE relation.

Default Values

As has already been discussed in Chapter 5 not every attribute value will necesarily be known when it becomes convenient to enter tuples in a relation. By definition an attribute which serves as a key may not accept default values. In certain other cases it might be benefical to the system to guarantee that an actual value is entered and specifically disallow default

values. Consider for instance the attribute NAME in PERSONAL. Clearly if the faculty's name is unknown it indicates that very little is known about that faculty member. Consequently in order to insure that enough information about the faculty member is availablle a new tuple will not be accepted unless a name is entered. In some cases attribute values are directly used in the scheduling process. For example the attribute ACTIVE in PERSONAL must be entered since the scheduling appliction programs will be applied to active facutly members only. A similar argument can be made about MAX#_CLASSES and DISCIPLINE in FACULTY, and any attributes dealing with times in SCHEDULE.

In one case an allowable default value will have a special meaning. Consider the SS# attribute in SCHEDULE. Until a class has been assigned to a faculty member the default value can signify that the class still needs to be assigned.

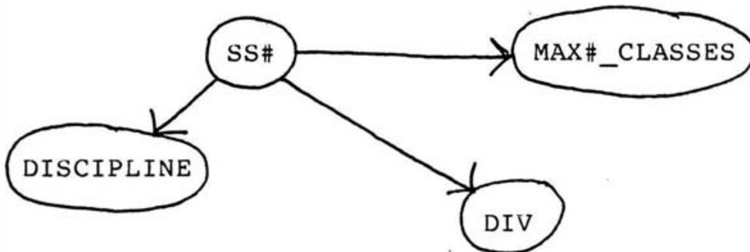
A formal presentation of the logical design of the data base is listed in APPENDIX A as part of the Data Dictionary. Included is the definition of each relation, the domains for each attribute in the relation including allowable default values and the integrity rules.

VERIFYING THE LOGICAL DESIGN

Recall that normal form theory provides a method by which the soundness of the data base's logical design can be verified. Without this verification unpredictable, inaccurate results could be generated through the use of the data base.

In order to apply normal form theory the relation's functional dependencies should be clearly defined. Each relation has been designed so that each nonkey attribute is functionally dependent on the key. Care must also be taken however to identify any dependencies that exist among the nonkey attributes.

Functional Dependencies in the Relations



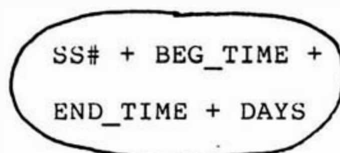
FUNCTIONAL DEPENDENCIES IN FACULTY RELATION

Figure 52

It could be argued that DISCIPLINE determines the DIVISION. A faculty member for instance whose

speciality is Pure Math is clearly associated with MAT division. Consider however the speciality Numerical Analysis. It could be that Computer Science and Mathematics faculty members have expertise in this field. Consequently the two attributes must remain independent.

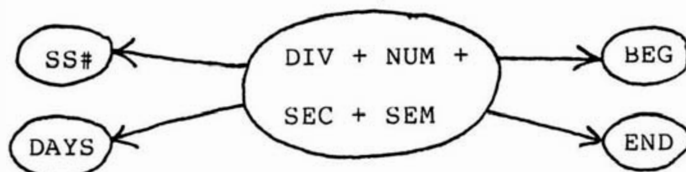
MAX#_CLASSES depends strictly on the faculty member's status each semester and is hence independent of all the other nonkey attributes.



NOT_AVAIL FUNCTIONAL DEPENDENCIES

Figure 53

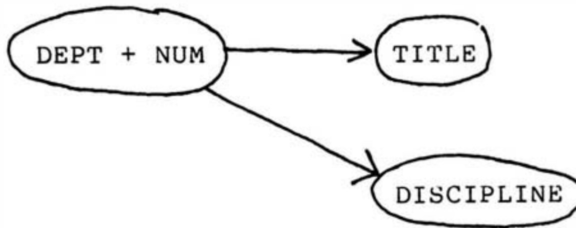
Since the entire tuple is the key of NOT_AVAIL there are no other functional dependencies to consider.



SCHEDULE FUNCTIONAL DEPENDENCIES

Figure 54

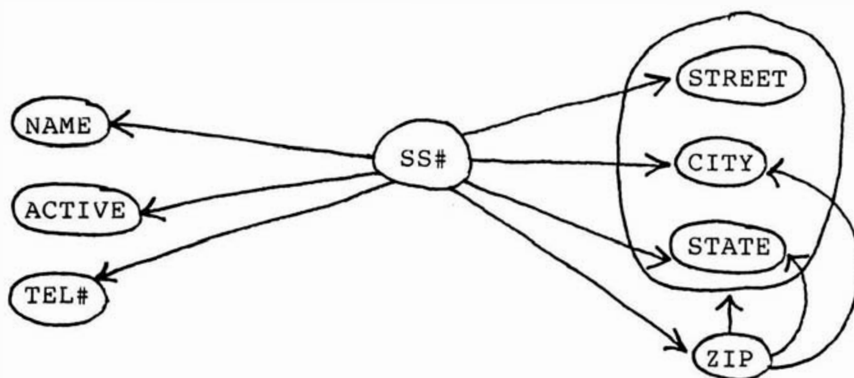
In many cases the days and the beginning time a class meets determines the ending time. For instance evening classes that start at 19:00 end at 20:15. There are however exceptions to these rules and consequently it must be assumed that these three attributes are independent. Also who teaches a course is generally independent of when that class meets. Therefore no dependencies exist among the nonkey attributes.



CLASSES FUNCTIONAL DEPENDENCIES

Figure 55

Since the title of a course may be ambiguous the course's discipline can not be inferred from it. TITLE and DISCIPLINE are therefore independent.



PERSONAL FUNCTIONAL DEPENDENCIES

Figure 56

In the case of PERSONAL there are definite dependencies among the nonkey attributes. Clearly a given zip code determines a state and also a city. The combination of Street, State and City determines a zip code as well. These nonkey dependencies could be avoided if the address was contained in one attribute. The flexibiility, however, of selecting on any particular one of these fields would be lost.

The ramifications of these nonkey attribute functional dependencies will be seen as the relations are tested against each of the five normal forms.

First Normal Form

By examining the data dictionary it can be verified that each attribute will only accept a single value from

its domain. A set of values for an attribute will not be accepted in any instance. Consequently all of the relations satisfy first normal form.

Second Normal Form

In the relations SCHEDULE and CLASSES which have multiple attribute values as the key it can be easily verified that the entire key is necessary in determing any nonattribute value. For instance in CLASSES the NUM value alone does not determine a class's title since multiple divisions could offer courses with the same NUM value.

All the relations satisfy second normal form.

Third Normal Form

By looking at Figure 56 it can be seen that STATE for instance is functionally dependent on the key as well as on ZIP. Consequently PERSONAL does not satisfy third normal form. All other relations however do not contain any functional dependencies among the nonkey attributes and hence do satisfy third normal form.

Boyce/Codd Normal Form

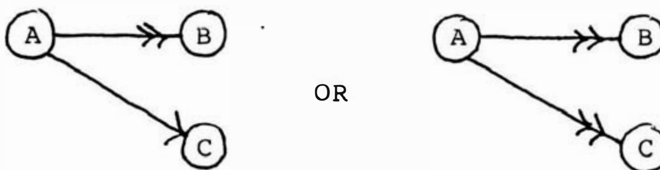
Since in all the relations excluding PERSONAL the only determinants are the actual keys of the relation they all satisfy Boyce/Codd normal form.

In PERSONAL however ZIP is a determinant since its

value will determine other attribute values. ZIP clearly is not a candidate key for this relation. PERSONAL then does not satisfy Boyce/Codd Normal form.

Fourth Normal Form

Recall that in order to have a multivalued dependency A->->B in R(A,B,C), there must be a set of B values matching a given (A,C) pair in R depending only on the A value and independent of the C value. Note



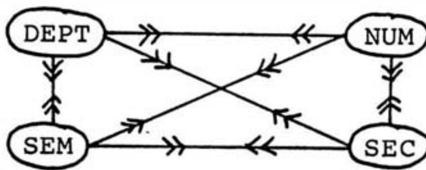
MULTIVALUED DEPENDENCIES

Figure 57

in Figure 57 that there is no relationship between the B and C attributes.

Since all of relationships from the key to a nonkey attribute are functional dependencies there are no multivalued dependencies originating from the key. Consider, however, relations where a set of attributes forms the key. Any multivalued dependencies between these attributes that form the key must be identified.

Since FACULTY has a single attribute as its key this relation has no multivalued dependencies and hence satisfies fourth normal form. The key of SCHEDULE is formed by the combination of the DIV, NUM, SEC, and SEMESTER attributes. The relationships among these attributes are shown in Figure 58.



KEY ATTRIBUTE RELATIONSHIPS IN SCHEDULE

Figure 58

Every attribute is dependent on every other attribute. For instance for each course number there will be many DEPT, SEC and SEM values associated with it. Clearly none of the attributes are independent of any others and no multivalued dependencies exist.

A similar argument can be made for CLASSES and NOT_AVAIL. Therefore all of the relations, excluding PERSONAL satsify fourth normal form.

Fifth Normal Form

Since in PERSONAL the ZIP determines the CITY and STATE this relation can be reconstructed by joining the

relations shown in Figure 59 over ZIP.

R

[SS# NAME STREET ZIP TEL# ACTIVE]

S

[ZIP CITY STATE]

POSSIBLE DECOMPOSITION OF PERSONAL

Figure 59

Since ZIP is not a candidate key for PERSONAL it fails to satisfy fifth normal form.

With respect to the remaining relations it is a nontrivial task to find all the join dependencies in the set of relations. Clearly there are several since at the very least each relation could be decomposed into a set of binary relations composed of the key and one nonkey attribute. Of course the key of each of these binary relations is the key of the original relation so these join dependencies do not imply that fifth normal form is violated.

When a relation is presenting interrelated multiple facts extra care should be taken in examining the join

dependencies. FACULTY is providing several pieces of unreleated information. In NOT_AVAIL the single piece of informtion presented by each tuple is a time period a faculty member is unavailable to teach. None of the attributes or combinations of them determine any remaining attribute values or combination of them. Similar arguments can be made concerning SCHEDULE and CLASSES.

It can be safely stated therefore that all the relations, excluding PERSONAL, satisfy fifth normal form.

PERSONAL's Failure to Satisfy the Normal Forms

It has been shown tht all of the relations excluding PERSONAL satsify the five normal forms. Their logical design has therefore been shown to be sound. Consider PERSONAL and the known abnormalities associated with its failure to satisfy the normal forms.

The entire problem with the PERSONAL relation revolves around the dependencies that orginate from the ZIP attribute. Abnormalities that could occur involve the zip code values. For instance if the zip codes were changed a sequential search would have to be made for every occurance of each code and each updated. Also if a faculty tuple was deleted it could be that a city

and zip code relationship will be deleted that is not listed anywhere else in the data base. These abnormalities are not a problem however. The chances that the zip code will be changed are few and there is really no value of having the data base contain a certain zip code city relationship. Consequently while these dependencies do in reality exist they can be ignored. It can be verified that if the dependencies originating from ZIP are eliminated PERSONAL satisfies all five normal forms.

THE PHYSICAL DESIGN

In order to design the physical level of the data base consideration had to be given to the computer system on which the data base management system will operate.

Currently at Virginia Commonwealth Univeristy an IBM 3081D is used. Batch and interactive processing are both available but the batch environment is much more widely used, more familiar and less costly. Also available is the WYLBUR interactive text editing procedure language. Through this command language many of the updating, inserting and deleting operations can be preformed interactively on sequential text files. The data base will operate in a batch environment but

simulate an interactive environment whenever possible. This interactive capability is very important in providing the most user friendly and flexible method of updating and modifying the files.

While the speed of operation is always a consideration when designing a data base system, in this case it will play a much smaller role. Generally the speed at which the interactive portion of the system will operate will depend most heavily on the number of users on the univeristy's computer system at the time an interactive command is given. The batch portion of the system will also depend on this factor since before a job is executed it must wait on a job queue, the size of which will depend on the number of users submitting jobs at that time. Consequently, the speed at which the operations are performed will depend heavily on factors outside the data base system's control.

The entire data base system will reside under a single account. There will be several users who can access this account but only one at a time. Therefore concurrent accessing of data is not an issue.

With respect to the actual design of the physical files that will be used, the most straighforward approach is to consider one physical file for each logical relation. In order to avoid confusion the same names will be used for both the logical relations and

their corresponding physical file. As has already been mentioned the response time of the data base system will depend on factors extraneous to the actual data base's design. It would be helpful however if the files were designed in such a way so that this problem would not be worsened by inefficient data structures. Several tests were therefore performed so that various data structures could be ranked with respect to the length of time it took to perform a set of given operations. A sample data collection was developed which contained 75, 80 character records (there are approximately 75 active faculty members) each containing a social security number, a name and an address. This sample data was then set up in various file designs and tested. Besides noting the time it took to set up the data structure two programs were run on each structure developed. The first searched for six records with specified social security numbers. The second searched for all records which contained a specified value in a nonkey field. In both cases the CPU time it took to execute the programs was noted. Not included in this time was any delay due to queueing.

Since each relation contains a key which distinguishes one tuple from any others a natural approach would be to design the file corresponding to the relations so that it indexed in some way on the key.

In the sample data the social seurity number could serve as the unique identifier for each record.

Initially an ISAM file was set up and tested. A major drawback to this structure is that on this particular operating system a minimum of one cylinder or 30 tracks of storage has to be allocated for each ISAM file. Clearly the 75 records in the sample data would not need that much space and in fact can reside on only one track.

Various hashing methods were then considered. A typical division on the key method, mod 79, was first tested for determining the record's address. A table of size 110 was chosen to allow for 45% excess space in order to provide for multiple keys mapping to the same location. Collisions were dealt with in several ways. First a simple linear insert was tested. A double hashing function was also tested where if on the second hash a collision occurred a linear insert was performed. The last test was to use a linked list to hold those records that collided at a specific index.

Another hash method which was also tested involved using the 4th and 5th digits of the social security number. Since these two digits were thought to be random they were used as the index into the table. Various collision methods were applied in this case

also.

Lastly a simple sequential file was tested. In this case no order was assumed and no direct accessing available.

The data in Table 2 summarizes the results of the tests run.

TABLE 2

COMPARISON OF EXECUTION TIMES IN SECONDS ON VARIOUS DATA STRUCTURES

FILE TYPE	FILE SETUP	SEARCH FOR 6 KEY VALUES	SEARCH FOR ALL OF NONKEY VALUE
ISAM	1.04	1.15	1.05
Sequential	.09	1.32	.99
Hashing - Division Method			
Linear Inserts	1.15	1.22	.92
Linked Inserts	1.16	1.58	.94
Double Hashed	1.19	1.36	.93
Hashing - 4th and 5th Digits of social security number			
Linear Inserts	1.20	1.24	.92
Double Hash	1.25	1.33	.93

While the ISAM file certainly performed marginally faster when searching on a key value the excessive space

required to hold the file clearly outweighs this slight advantage. Another disadvantage is an ISAM file's inability to be accessed dircectly by the interactive WYLBUR procedures. Therefore ISAM files were not considered further.

With respect to the hashing methods used versus the sequential design there is really not much difference in the times shown. This is because the usual advnatages the hashing tables provide do not become apparent until the files searched become much larger. Since the size of files in the data base will generally be small this was an important consideration.

Besides being more difficult to program and therefore more prone to error the linked list method of inserting in a hash table provided no advantage in searching times. Similarly the double hashing method did not show any advantage by its use. These two methods were therefore also no longer considered. Note also that the two digit method of hashing provided no improvement over the division method. Since the division method is the more straightforward the two digit method was dropped from further consideration.

While the sequential file structure does indeed

require slightly longer search times then the hashing method there are some advantages of this design to consider. Clearly of the two the sequential design is the simplest to program and maintain. While WYLBUR would indeed be able to access either file structure, in the case of the hashed file, each time an interactive insertion session is completed a batch program would have to be run to insure that any records inserted are placed at their correct address. WYLBUR provides no straightforward mechanism for determining a record's correct hashed location and then inserting it there. This extra program execution would cause an undesirable complication to the interactive processing. Inserting a record into a sequential file however would merely require writing it to the end of the file.

In order to make a decision on the file structures chosen consideration was given to the types of activity for which the user would require the quickest response time. Clearly the interactive portion must be the quickest since when submitting a batch job a delay is expected. Also since users will generally be inexperienced with the computer the interactive sessions must be as straightforward as possible. Therefore in order to meet these criteria and since the time loss is not significant the sequential design was chosen as the

initial file structure for the data base. As the data base grows these structures can certainly change in order to provide the most efficient system.

For each logical relation a physical sequential file will be built. In most cases the record definition will mirror the relation's tuple definition. Of course the particular field definition must be much more precise. For instance the field DISCIPLINE must be limited to say 20 characters. Recall, however, that it is not necessary for the record's field value to match the logical attribute value. Consider the BEG_TIME and END_TIME field. In most cases comparisons over a range will be made on these fields. A search might be made for instance for faculty members who are available from 10:00 to 11:00 on specified days. Clearly it would be difficult to consider a range of times on these 5 character values. Therefore any times stored will be converted to an integer representing the number of minutes from midnight for each time. 10:00 then would be stored as 600. The user of course will be oblivious to this conversion and therefore the data manipulation language and any application programs will take this connversion into account. In order to make the best use of the WYLBUR interactive capabilities all data will be stored as character strings and converted to integers

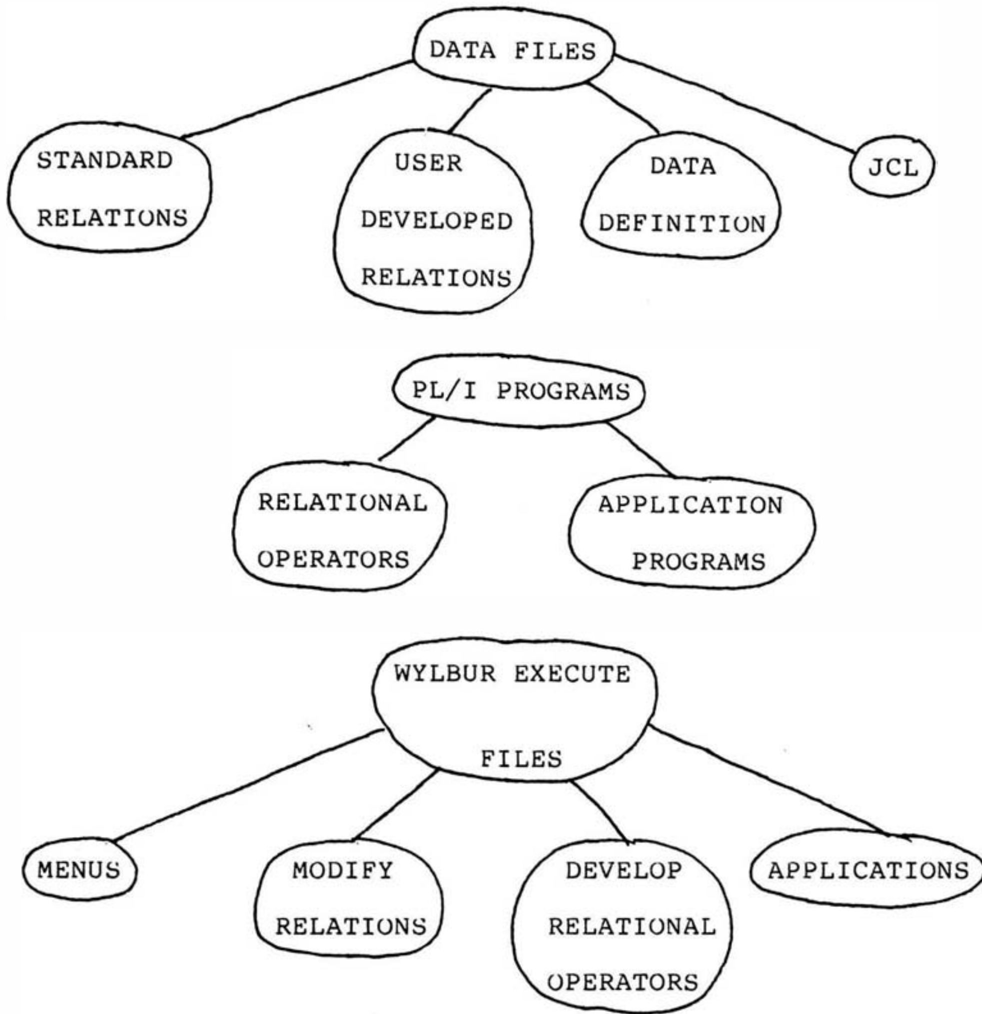
when necessary.

Listed in the Data Dictionary in Appendix A is the actual definition of the physical data base.

DESIGN IMPLEMENTATION

The relational data base described in the previous sections will be implemented on one account through a series of menu driven procedures. The user need only be familiar with the logical view of the data base and the relational operators. The WYLBUR command language will be used to generate the interactive menus and also to allow the user to interactively update relations and build programs to perform the relational operations in a batch mode. The actual programs will be written in PL/I for its ease of file accessing. Various files will reside on this account and can be categorized as either data files, WYLBUR execute files or PL/I program files (Figure 60).

Included in the set of data files will be the physical sequential files representing the relations composing the logical view of the data base. These will be referred to as the standard relations. Of course the user should have the ability to generate new relations and save them for later reference so for each of these user developed relations a physical sequential file will be generated. Now in order to clearly define the physical record representing the logical tuple in each relation a file will be maintained which will include for each standard and user developed relation a PL/I



CONTENTS OF DATA BASE ACCOUNT

Figure 60

record definition. These record definitions will be copied by WYLBUR into PL/I programs which will access the corresponding file.

The WYLBUR command language files can be classified into the groups shown in Figure 60. Menus will be used to prompt the user for courses of action from the time the user logs on to when he logs off. For each standard relation a WYLBUR execute file will allow the user to modify the relation. The user will be allowed to insert, delete, or update tuples within the constraints defined by the data dictionary. Another WYLBUR execute file will prompt the user for various portions of the scheduling problem. This WYLBUR execute file is really separate from the relational data base design and can be considered the first of what could be several application oriented WYLBUR exec files. The last set of WYLBUR execute files noted will allow the user to build PL/I programs which will perform any relational operation desired. Of course the user will be oblivious to the PL/I aspect of this operation and will only answer a series of prompts which will determine the PL/I code WYLBUR will generate and then insert into a core PL/I program.

Many of the PL/I programs run will be developed by WYLBUR and built from the core program. This core program will contain a constant set of JCL, a subroutine which will print generated relations and a call to this

subroutine. Another file will contain the outlines of the subroutines which will perform the relational operations. A set of application programs pertaining to the scheduling system will also initially reside in the account and will be run through WYLBUR execute files.

In order to develop algorithms for the various portions of the system it would be helpful to consider the flow of control as the user logs on and proceeds through the various options available.

Logging On

Whenever any user logs on to this account the

```
WELCOME TO THE MATH SCIENCES DATA BASE SYSTEM

                    OPTIONS
   A.  MODIFY A STANDARD RELATION
   B.  BUILD A NEW RELATION
   C.  DELETE A NONSTANDARD RELATION
   D.  APPLICATIONS
   E.  SCHEDULING SYSTEM
   F.  LOG OFF
ENTER ONE OF THE ABOVE OPTIONS -->
```

LOGIN MENU

Figure 61

system will automatically execute a login command file. This command file then will present the main menu shown in Figure 61 and prompt the user for a course of action.

Consider now the implementation of each of the options listed.

Modify a Standard Relation

If the user enters A a new menu will be presented listing each of the standard relations and prompting the user for which relation is to be modified. Control will then be passed to the appropriate command file which will allow the user to modify that relation. There will be one procedure for each standard relation. The algorithm listed in Appendix B defines the execute file which will allow the user to modify the PERSONAL relation.

Note that the integrity rules listed in the data dictionary are all enforced. For instance, if the ACTIVE value becomes no through an update any records in FACULTY or NOT_AVAIL are deleted and SS# in SCHEDULE is set to blank.

The WYLBUR procedures to modify each of the other standard relations will be similar.

Build a New Relation

The user has the option to use one of the relational operators to build a new relation. Only one operator can be used at a time but the user has the option of saving the generated relation and then applying another operator on that relation. This process can be repeated allowing the user to apply several operators.

As the user answers various prompts an executable program which will perform the desired operation will be built within the core program described earlier. The appropriate subroutine will be copied into the core and then updated as the user answers various prompts. Record definitions for the necessary standard relations will be copied from the data defintion file into the core and a record definition for the operand relation built. JCL for the operand standard relation files and the output relation file will be put into the core. A partially built program which will perform a select is shown in Figure 62. Capital letters indicate a fixed PL/I command and small letters indicate code that will be updated with information provided from the user through prompts.

Appendix B lists algorithms for the WYLBUR command file which will build the executable SELECT program. Also included in this appendix are the algorithms used to perform the relational operators.

```
//jcl                          (to run PL/I program)

 CORE:   PROCEDURE OPTIONS (MENU);
         DECLARE

           1  in_recl
           1  out_rec
           file declarations
           SYSPRINT FILE STREAM;

           CALL SELECT;
           CALL PRINT_OUT;

 SELECT: PROCEDURE;
          DECLARE
             EOF BIT(1);
          EOF='0'B;
          ON ENDFILE (infile)EOF='1'B;
          OPEN FILE (infile) INPUT;
          OPEN FILE (outfile) OUTPUT;
          READ FILE (infile) INTO (in_recl);
 LOOP:    DO WHILE (   EOF);
             IF (condition) THEN
                WRITE (outfile) out_rec;
             READ FILE (infile) INTO (in_recl);
          END LOOP;
          CLOSE FILE (infile);
          CLOSE FILE (outfile);
          END SELECT;

 PRINT_OUT: PROCEDURE:
             DECLARE
                EOF BIT(1);
             EOF='0'B;
             OPEN FILE (outfile) INPUT;
             ON ENDFILE (outfile) EOF='1'B;
             READ FILE (outfile) INTO (out_rec);
 LOOP:       DO WHILE (   EOF);
                PUT (
             END LOOP;
             CLOSE FILE (outfile);
             END PRINT_OUT;

        END CORE;

//jcl                        (defining files used)
```

SAMPLE PL/I PROGRAM

Figure 62

Delete a Relation

The user may decide to delete any relations that have been generated. Of course this does not include the standard relations. The user will be prompted for a nonstandard relation to be deleted. The corresponding file will then be scratched.

Applications

If the user enters this option a menu will be presented showing application programs available. This list will vary with time but will initially contain an option that will allow the user to request hard copies of the data in each of the standard relations.

The user will be prompted for the relation to be printed. A corresponding PL/I program will then be submitted which prints the file with appropriate headings and titles.

Scheduling System

This set of WYLBUR execute files and PL/I programs will mechanize the scheduling system described at the beginning of this chapter. The process will mirror the steps currently being handled manually.

The algorithms listed in Appendix B outline this

process.

Logging Off

The user will enter this option when the data base session is complete. The account will then be logged off.

Another, hidden, option will be available for the data base administrator and application programmers to use. If this option is entered the programmer will exit from the login menu and any WYLBUR execute files and have access to all levels of the account.

CONCLUSION

CHAPTER 8

The Department of Mathematical Sciences Scheduling data base presented in Chapter 7 illustrates the use of a data base environment through which a given problem can be solved. In particular the use of a relational data base system provides a logical design even the most inexperienced user can understand and easily access.

While the implementation of this system will take longer than a system designed to address only the given problem its use will be much more flexible. By simply quering the system, for instance, a mailing list of all active faculty members can be generated. The data maintained by the system then can be used in numerous unanticipated ways making optimum use of the information maintained.

Future expansion of the logical design can also be

easily made. Suppose information was needed concerning the text books used by each course. A relation would be designed which maintained these textbook titles using the course's number and division to identify the course. The data manipulation language would be expanded to provide for this new relation but existing applications would continue to run without modification.

In addition, by maintaining all the department's information in the data base system the danger of inconsistent data is controlled and therefore minimized. Any results derived from the system can be considered accurate.

A relational data base environment then should be a consideration when designing a computer system to solve a recurring problem that requires interrelated data.

APPENDIX A

LOGICAL DATA DICTIONARY

PHYSICAL DATA DICTIONARY

BIBLIOGRAPHY

Aho, A. V., C. Berri, and J. D. Ullman, "The Theory of Joins in Relational Databases" ACM Transactions on Database Systems, Vol. 4, #3, September, 1979, pp. 297-314.

Codd, E. F., "A Relational Model of Data for Large Shared Data Banks" Communications of ACM, Vol. 13, #6, June, 1970, pp. 377-387.

Codd, E. F., "Rational Database: A Practical Foundation for Productivity" Communications of ACM, Vol. 25, #2, February, 1982, pp. 109-117.

Codd, E. F., "Extending the Database Relational Model to Capture More Meaning" ACM Transactions on Database Systems, Vol. 4, #4, December, 1979, pp. 397-434.

Date, C. J., An Introduction to Database Systems, Volume II, Reading, Massachusetts: Addison-Wesley Publishing Company, 1983.

Date, C. J., An Introduction to Database Systems, 3d ed, Reading, Massachusetts: Addison-Wesley Publishing Company, 1982.

Kent, William, "A Simple Guide to Five Normal Forms in Relational Database Theory" Communications of ACM, Vol. 26, #2, February, 1983, pp. 120-125.

Martin, James, An End-User's Guide to Data Base, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1981.

Martin, James, Managing the Data Base Environment, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1983.

Martin, James, Computer Data Base Organization, 2d ed., Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1977.

Ullman, Jeffrey D., Principles of Database Systems, Rockville, Maryland: Computer Science Press, 1982.

PERSONAL

SS#	NAME	STREET	CITY	STATE	ZIP	TEL#	ACTIVE

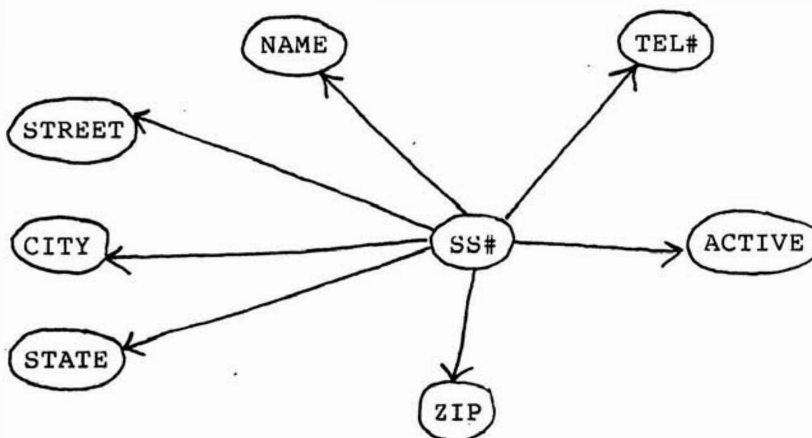
ATTRIBUTES:

SS#	Social security number of faculty member Default: none
NAME	Last name, First name Default: none
STREET	Street address of residence Default: Dept. of Math. Sciences VCU
CITY	City of residence Default: Richmond
STATE	Two letter state abbreviation of residence Default: VA
ZIP	5 digit zip code of residence Default: 23349
TEL#	Residence telephone number Default: 257-1301
ACTIVE	Yes or No Default: none

INTEGRITY RULES:

1. If ACTIVE is Yes then tuple must be included in FACULTY with same SS#.
2. If SS# updated then SS# must be updated in FACULTY and SCHEDULE.
3. If ACTIVE becomes No then tuples in FACULTY and NOT_AVAIL deleted.

FUNCTIONAL DEPENDENCIES



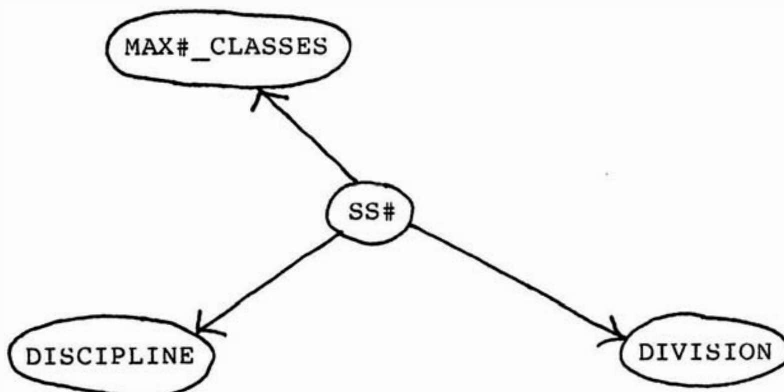
FACULTY

SS#	MAX#_CLASSES	DIVISION	DICIPLINE

ATTRIBUTES:	SS#	Social security number of faculty member Default: none
	MAX#_CLASSES	Integer >= 0 Default: none
	DIVISION	MAT, STA, or CSC Default: none
	DISCIPLINE	Allowable speciality Default: Division value

INTEGRITY RULES: 1. For each tuple included there must be a corresponding tuple in PERSONAL with same SS# and ACTIVE Yes.

FUNCTIONAL DEPENDENCIES



NOT_AVAIL

SS#	BEG TIME	END TIME	DAYS

ATTRIBUTES:

SS#	Social security number of faculty member Default: none
BEG_TIME	Beginning time on army clock when faculty member not available Default: none
END_TIME	End time on army clock when faculty member not available Default: none
DAYS	MTWRF where letter is blank when day not applicable Default: none

FUNCTIONAL DEPENDENCIES

SS# + BEG_TIME

END_TIME + DAYS

SCHEDULE

<u>DIV</u>	<u>NUM</u>	<u>SEC</u>	<u>SEMESTER</u>	BEG	END	DAYS	SS#

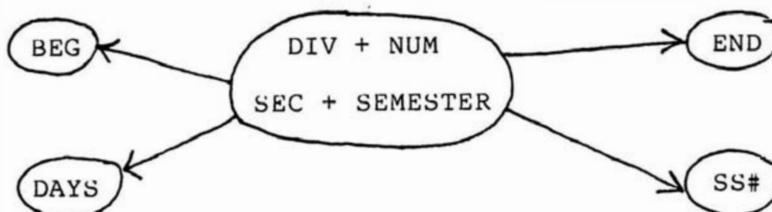
ATTRIBUTES:

DIV	MAT, STA, or CSC Default: none
NUM	Three digit catalog number of course. Default: none
SEC	Three digit section number followed by E if evening or blank if day class Default: none
SEMESTER	Spring, Summer or Fall Default: none
BEG	Army time of when class begins Default: none
END	Army time of when class ends Default: none
DAYS	MTWRF when days class does not meet are blank. Default: none
SS#	Social security number of faculty member teaching course. Default: ' '

INTEGRITY RULES:

1. For each DIV and NUM a tuple must be included in CLASSES with same DIV and NUM values.
2. Every SS# value must be listed in FACULTY and PERSONAL with an ACTIVE value of Yes.

FUNCTIONAL DEPENDENCIES



CLASSES

DIV	NUM	TITLE	DISCIPLINE

ATTRIBUTES:

DIV	MAT, STA or CSC Default: none
NUM	Three digit catalog number of class. Default: none
TITLE	Title of course as listed in catalog. Default: none
DISCIPLINE	One of allowable disciplines. Default: Div value

INTEGRITY RULES:

1. If DIV or NUM updated corresponding values updated in SCHEDULE.
2. If tuple deleted then tuples in SCHEDULE with same NUM DIV values must be deleted.

FUNCTIONAL DEPENDENCIES

TITLE ← DIV + NUM → DISCPLINE

PERSONAL

FILE TYPE: Physical Sequential

RECORD DEFINTION:

```
PERSON
  SS#        Character 11 ddd-dd-dddd
  NAME       Character 25 last name, first name
  STREET     Character 20
  CITY       Character 15
  STATE      Character  2
  ZIP        Character  5 ddddd
  TEL#       Character  8 ddd-dddd
  ACTIVE     Character  3 Yes or No
```

FACULTY

FILE TYPE: Physical Sequential

RECORD DEFINITION:

```
TEACHER
  SS#            Character 11 ddd-dd-dddd
  MAX#_CLASSES   Character  1 integer value 0 -9
  DIVISION       Character  3 MAT, STA or CSC
  DISCIPLINE     Character 20
```

NOT_AVAIL

FILE TYPE: Physical Sequential

RECORD DEFINITION:

```
TIME
  SS#        Character 11 ddd-dd-dddd
  BEG_TIME   Character  4 0 - 1440
  END_TIME   Character  4 0 - 1440
  DAYS       Character  5
```

SCHEDULE

FILE TYPE: Physical Sequential

RECORD DEFINITION:

```
SECTION
   DIV         Character  3 MAT, STA or CSC
   NUM         Character  3 integer 100 - 999
   SEC         Character  4 integer 1 - 99
                   followed by ' ' or 'E'
   SEMESTER    Character  2 SP, FA, or SU
   BEG         Character  4 0 - 1440
   END         Character  4 0 - 1440
   DAYS        Character  5
   SS#         Character 11 ddd-dd-dddd
```

CLASSES

FILE TYPE: Physical Sequential

RECORD DEFINTION:

```
CLASS
   DIV          Character  3 MAT, STA or CSC
   NUM          Character  3 integer 100 - 999
   TITLE        Character 40
   DISCIPLINE   Character 20
```

APPENDIX B

IMPLEMENTATION ALGORITHMS

In order to be consistent with the WYLBUR practice of using labels as line designators L1, L2, etc. will be used in a similar way.

ALGORITHM TO MODIFY PERSONAL RELATION

```
L1:  Present Personal Edit Menu
         A.  Insert a new faculty member
         B.  Delete a faculty member
         C.  Update a faculty member's data
         D.  Return to Main Menu

     Prompt for option

     If user enters option to insert faculty member
        Prompt for social security number - SS#
        Search PERSONAL for SS#
        If record not found then
           Prompt for last name
           If last name not blank then
              Prompt for first name, street, state, zip,
                 telephone
              Prompt user if Active or not
              If not active then
L2:              Search FACULTY for SS#
                 If found then delete record
                 Search NOT_AVAIL for SS#
                 While found delete record
                 Search SCHEDULE for SS#
                 While found set SS# in SCHEDULE to ' '
              else if active then
                 Build record for PERSONAL
                 Write record to file PERSONAL
L3:              Prompt for MAX#_CLASSES, DIVISION,
                    DISCIPLINE
                 Build record for FACULTY
                 Write record to file FACULTY
           Goto L1

     If user enters option to delete faculty member
        Prompt user for last name, first name
        Search PERSONAL for that record
        If found then
           If correct record then
              Delete record from PERSONAL
              If ACTIVE value was Yes then
                 Goto L2
```

```
If user enter option to update a record then
   Prompt user for last name
   Search PERSONAL for record
   If record found then
      If correct record then
         Allow user to update information
         Build record
         If SS# changed then
            Search PERSONAL for new SS#
            If record not found then
               Write record to PERSONAL
            Update SS# in FACULTY, NOT_AVAIL,
                 SCHEDULE
         If ACTIVE changed then
            If now not active then
               Goto L2
            If now active then
               Goto L3

If user enters option to return to main menu then
   Execute Login Menu
```

ALGORITHM TO EXECUTE SCHEDULING SYSTEM

```
     Prompt user for semester they are working on

     Prompt user if they are beginning a new schedule

     If a new schedule then
        Set ss# to blank in SCHEDULE for all records
           with that semester value

L1:  Present Scheduling System Menu
        A.  Print listing of current schedule
        B.  Assign faculty to class
        C.  Change classes scheduled
        D.  Print possible faculty assignments to
              unassigned classes
        E.  Return to Main Menu

     If user enters option to print current schedule
        Edit PL/I program PRINTER so that semester
           desired is printed.
        Submit PL/I program PRINTER that prints schedule
              in grid format.
        Print message to user to pick up printout.
        Execute Login Menu

     If user enters option to assign faculty to classes
        Prompt user for class division, number and
              section
        Form key for SCHEDULE with input information
        Search for class record
        If record found then
L5:        Prompt user for faculty name
           Search PERSONAL for name
           If record found then
              If correct record then
                 If Active value in PERSONAL is Yes then
                    Write SS# from PERSONAL record to
                          SCHEDULE record
                    Write updated record to SCHEDULE
        Goto L1

     If user enter option to change schedule then
        Present Menu
           1.  Delete a class
           2.  Insert a class
           3.  Return to Scheduling Menu
```

```
   If user enters option to delete class then
      Prompt user for class number, division and
            section
      Search SCHEDULE for class
      If class it found then
         Delete class

   If user enters option to insert a class then
      Prompt user for class number, division and
            section
      Prompt user for times and days class meets
      Build SCHEDULE record
      Prompt user if class is assigned
      If class is assigned then
         Goto L5
      If class is not assigned then
         Write record to SCHEDULE

   If user enters option to return to menu then
      Goto L1

If user enters option to print possible assignments
   Edit PL/I program GENERATE with semester value
   Submit GENERATE which will find and print
         possible faculty assignments noting any
         that are better choices according to
         scheduling criteria.
   Print message to user to pick up printout.
   Exec Login Menu

If user enter option to return to Main Menu
   Exec Login Menu
```

IMPLEMENTATION ALGORITHMS FOR THE RELATIONAL OPERATORS

Let R, S denote physical files corresponding to relations $R[A_1,\ldots,A_n]$ and $S[B_1,\ldots,B_m]$ respectively.

Let R.AI denote the field of the physical record defintion corresponding to A_i attribute of R.

Let T denote a physical file corresponding to output relation T.

SELECT T = R Where $A_i = v$

Record definition for T same as for R

```
While more records in R
   Read record in R
   If R.AI = v then
      Write record to T
End
```

PROJECTION $T[A_i,A_j,\ldots,A_k] = R[A_i,A_j,\ldots,A_k]$

T's record defintion will be composed of those fields from R corresponding to attributes projected from R.

```
While more records in R
   Read record in R
   T.AI = R.AI
   T.AJ = R.AJ
   ...
   T.AK = R.AK
   Write new record to T
End
```

```
UNION    T = R UNION S


         Record definition for T same as for R and S

         While more records in R
            Read record in R
            Write record to T
         End
         While more reocrds in S
            Read record in S
            Found is false
            While more records in T and found is false
               If records are equal found is true
            End
            If not found
               Write record from S to T
         End


INTERSECTION   T = R INTERSECT S

         Record definition for T same as for R and S

         While more records in R
            Read record from R
            Found is false
            While more records in S and found is false
               Read record from S
               If records are equal
               Write record to T
               Found is true
            End
         End


MINUS    T = R MINUS S

         Record definition for T same as for R and S

         While more records in R
            Read record in R
            Found is false
            While more records in S and found is false
               If records are equal found is true
            End
            If found is false
               Write record from R to T
         End
```

CROSS $T[A_1,\ldots,A_n,B_1,\ldots,B_m] = R[A_1,\ldots,A_n]$ CROSS $S[B_1,\ldots,B_m]$

Record definition for T will be composed of all fields from R and S.

```
While more records in R
   Read record in R
   While more records in S
      Read record in S
      T.A1 = R.A1
      ...
      T.An = R.An
      T.B1 = S.B1
      ...
      T.Bm = S.Bm
      Write new record to T
   End
End
```

JOIN $T[A_1,\ldots,A_j,\ldots A_n,B_1,\ldots,B_{k-1},B_{k+1},\ldots B_m] =$

R Join S where the join is over the attribute A_j, B_k subsets of the same domain.

Record definition for T will be composed of all fields from R and S excluding the B_k attribute from S.

```
While more records in R
   Read record in R
   While more records in S
      Read record in S
      If R.AJ = S.BK then
         T.A1 = R.A1
         ...
         T.An = R.An
         T.B1 = R.B1
         ...
         T.Bk-1 = S.Bk-1
         T.Bk+1 = S.Bk+1
         ...
         T.Bm = S.Bm
         Write new record to T
   End
End
```

Vita