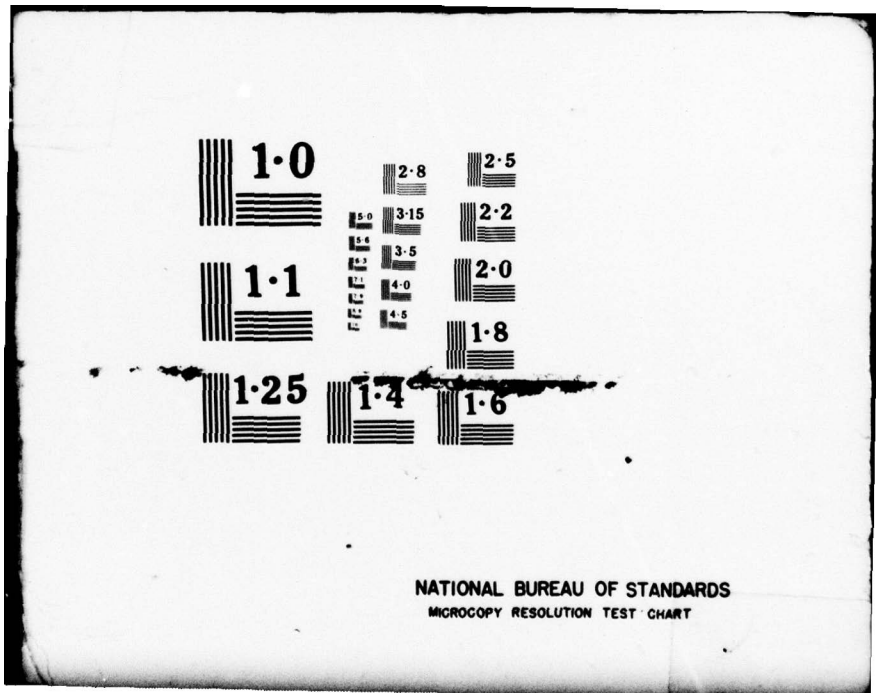
AD-A057 902 NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF F/G 9/2
LOGICAL DATA BASE DESIGN FOR RELATIONAL DATA BASE SYSTEMS.(U)
JUN 78 J A CHAPMAN

UNCLASSIFIED NL

| OF |
ADA
057902

END
DATE
FILMED
10-78
DDC


1·0
1·1
1·25
2·8
3·15
3·5
4·0
4·5
1·4
2·5
2·2
2·0
1·8
1·6
NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

# NAVAL POSTGRADUATE SCHOOL
## Monterey, California




(9) Master's THESIS,

(6) Logical Data Base Design for Relational Data Base Systems,

by

(10) Jack Albert/Chapman

(11) June 1978

(12) 62 p.

Thesis Advisors: S. T. Holl
N. F. Schneidewind

Approved for public release; distribution unlimited.

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle)<br>Logical Data Base Design for Relational Data Base Systems | | 5. TYPE OF REPORT & PERIOD COVERED<br>Master's Thesis; June 1978<br>6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s)<br>Jack Albert Chapman,<br>Lieutenant Colonel, USMC | | 8. CONTRACT OR GRANT NUMBER(s) |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS<br>Naval Postgraduate School<br>Monterey, CA 93940 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS |
| 11. CONTROLLING OFFICE NAME AND ADDRESS<br>Naval Postgraduate School<br>Monterey, CA 93940 | | 12. REPORT DATE<br>June 1978<br>13. NUMBER OF PAGES |
| 14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)<br>Naval Postgraduate School<br>Monterey, CA 93940 | | 15. SECURITY CLASS. (of this report)<br>UNCLASSIFIED<br>15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

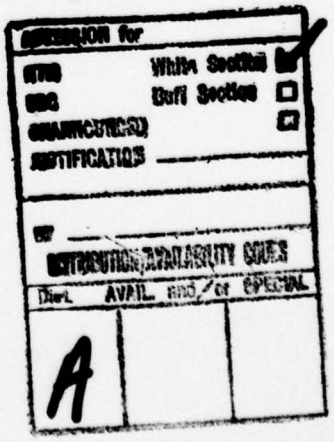
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

logical data base design
data base management systems

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Theories and guidance for logical data base design have lagged far behind the advances in physical data base design. The advent of sophisticated data base management systems has relieved the user of many of the problems of physical data base design, but has placed more emphasis on good logical design. The theory behind logical data base design is → next page

explored and a step by step procedure for the logical data base design is presented, employing mathematical and operations research techniques.



Approved for public release; distribution unlimited.

# Logical Data Base Design

# for

# Relational Data Base Systems

by

Jack Albert Chapman
Lieutenant Colonel, United States Marine Corps
B. S., Illinois Institute of Technology, 1961

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1978

Author Jack Albert Chapman

Approved by: Stephen T. Toll
Thesis Advisor

Norman F. Schneidewind
Co-Advisor

Chairman, Department of Computer Science

Dean of Information and Policy Sciences

## ABSTRACT

Theories and guidance for logical data base design have lagged far behind the advances in physical data base design. The advent of sophisticated data base management systems has relieved the user of many of the problems of physical data base design, but has placed more emphasis on good logical design. The theory behind logical data base design is explored and a step by step procedure for the logical data base design procedure is presented, employing mathematical and operations research techniques.

TABLE OF CONTENTS

# I. INTRODUCTION

Data base technology has received considerable attention in recent years. Much research on data bases has been directed toward exploring the problem of physical data base design and how the data within the data base will be stored for subsequent accessing. The author suspects that the heavy emphasis on the physical data base design aspect can be attributed to the quest for software methods of integrating already well established file management methods.

Physical data base design deals with how to best use alternative storage methods such as sequential files, indexed sequential files, linked lists, etc., to enhance retrievability of stored data while keeping storage and software overhead within acceptable limits.

The advent of data base management systems (DBMS) has been the result of considerable research done in this area. With a data base management system, the user is relieved of much of the task of physical data base design. The user, or implementor, must now be much more concerned with logical data base design in order to make good use of the capabilities of the data base management system.

Logical data base design deals with how to conceptually structure the data to meet the needs of the user community and to efficiently fit it within the framework of the physical data base design.

Unfortunately, research on logical data base design has lagged far behind research on physical data base design. (The

analogy between this problem, and the gap between computer hardware engineering and computer software engineering is probably no accident.) The problems of physical data base design lend themselves to more precise definition than do the problems of logical data base design.

The purpose of this thesis is to explore the factors involved in logical data base design as they apply to relational data base systems, and to devise a step by step logical data base design procedure.

Appendix A contains a brief description of the concept of relational data base systems.

## II. NATURE OF THE PROBLEM

The entire data base design process has been described as consisting of the following phases (Ref 2):

Phase 1 - Information and requirements specifications and analysis.

Phase 2 - Logical data base design.

Phase 3 - Evaluation of logical data base design(s).

Phase 4 - Physical data base design.

Phase 5 - Evaluation of physical data base design(s).

Phase 6 - Data base construction and initialization.

Phase 7 - Performance evaluation.

As a result of Phase 1, the designer of a data base will have analyzed the information requirements of the organization and will have identified the key entities within the organization. He will have further identified those data items of each entity that must be processed by the organization.

An organization's data is described in terms of entities. An entity is a real world object such as an employee, a product, a department, or a student; or it may be an abstract entity such as a course or a billing. Entities have properties that describe the entity, or identify the entity, or represent relationships between the entity and the real world. (Ref 2)

In the realm of considering information about entities, the properties are referred to as attributes. (Ref 3)

The designer's task in Phase 2 is to identify interconnections between entities and to select groupings of attributes for each

entity into relations. The groupings of attributes into relations must be such that they will best suit the needs of the organization and will be "efficient". (The question of what is "efficient" will be addressed later.)

A chief benefit of an integrated data base is being able to access information on the relationships between entities. (What department does Jones work in? How many widgets were produced last month and shipped to warehouse number 3? etc.) This type of information accessing requires not only the grouping of attributes of a given entity, but also the inclusion of attributes of other entities so that links can be made. These attributes of other entities are referred to as "foreign keys". The foreign key grouped with the attributes of one entity allows the establishment of relationships between that entity and the entity to which the foreign key belongs.

When there exist only a small number of entities, each having a small number of attributes (such as the example in Appendix A), the logical design process is relatively simple and can be readily accomplished by purely intuitive means.

When, however, there are entities with a large number of attributes, the problem of grouping attributes into relations becomes considerably more complex. Given one entity with k attributes, there are nearly 2 to the kth power possible ways to group the attributes into relations (that, of course, would include the possibility of only one attribute per relation, which would not be very useful). Add to that, two other entities with 1 and m attributes respectively, and the number of possible

groupings within the data base approaches 2 to the $(k+l+m)$ power.

The designer of the logical data base must have means available to drastically reduce the number of possible ways in which he can develop the logical data base.

## III. BASIC CONSIDERATIONS AND DATA BASE TERMINOLOGY

It is appropriate at this point to identify some of the basic, theoretical aspects of entities, attributes, relations and the assignment of attributes to relations.

### A. RULING PART - DEPENDENT PART

Each tuple within a relation (a row in the two-dimensional table) is composed of a ruling part and a dependent part. The ruling part is the key that uniquely identifies that particular tuple. The ruling part (key) must be nondecomposable - the ruling part can not contain one or more attributes that could be deleted, still allowing the ruling part to uniquely identify the tuple. The key can be, and often must be, composed of two or more attributes. It is possible to have relations in which all attributes are part of the key.

### B. FUNCTIONAL DEPENDENCE

If the value of an attribute B is always determined by the value of attribute A, then B is functionally dependent on A (an employee's address is functionally dependent on his name).

### C. RELATIONS

There are 5 basic types of relations that can be used within a relational data base (Ref 4):

An ENTITY RELATION defines a set of entities. For example:

STUDENT-INFO( [STUDENT-NAME], SSN, AGE, MAJOR-NUMBER)

(When relations are given in narrative form, the [ ] enclose the key.) It is the most fundamental type of relation. Updating of a tuple of another relation containing information about the student will not require updating a tuple of the entity relation.

An update (or deletion) of a tuple in the entity relation may require updating tuples in one or more other relations.

A LEXICON RELATION is based on a one-to-one correspondence between two attributes where either of the attributes can be the ruling part. An example of a lexicon relation is:

IDENT( [NAME], SSN)

The SSN is functionally dependent on name, but name is also functionally dependent on SSN. The existence of lexicons in the data base can be helpful in establishing keys for referenced relations.

A REFERENCED RELATION contains tuples that are referenced by a dependent attribute(s) in an entity relation. In the entity relation example, there was a MAJOR-NUMBER. That MAJOR-NUMBER can be used as a key to access more information about a particular major such as in the following referenced relation:

SUBJ( [MAJOR-NUMBER] ,SUBJECT,DEPT)

Weiderhold (Ref 4) supplies a note of caution regarding updating of referenced relations:

"The modification of a referenced relation is restricted; tuples may not be removed while any reference to the tuple exists. The removal of a reference, however, does not imply removal of the referenced tuple."

An ASSOCIATIVE RELATION contains data relevant to the relationships of entity relations. The ruling part, or key, is composed of two or more attributes. Those attributes relate tuples in the associative relation to tuples in two or more owner relations.

Suppose that a data base has data on students and courses.

A relation giving time and place of final examination would be an example of an associative relation.

FINAL( [STUDENT-NAME,COURSE],TIME, LOCATION)

The two attributes in the key point back to relations giving data about students and courses. Associative relations are important for representing data common to two or more entities.

The final type of relation is the NEST RELATION. Nest relations contain tuples of repeating groups. A nest relation would be required to represent all of the classes presently being taken by a student:

ENROLL( [NAME,COURSE], CREDIT-HOURS)

The key of a nest relation is a composite of the key of the entity relation referencing the nest relation (NAME), and an attribute within the nest relation that uniquely identifies the tuple (COURSE). The composite key is necessary because for a student with n courses, n tuples in the relation will include the student's name. Similarly, if m students are taking a course, that course name will appear in m different tuples.

D. ATTRIBUTES

Kahn (Ref 2) has identified 5 different types of attributes.

An attribute can uniquely identify an entity.

An attribute can describe an entity (age, rank, etc.).

An attribute can represent relationships between entities (a course taken by a student describes a relationship between those two entities).

An attribute can represent derived properties that can be calculated (age = current date - date of birth).

Finally, an attribute can be a property "invented" to assist in the manipulation of a data base.

E. NORMAL FORMS

The designer of a logical data base must understand the concepts of "normal forms". The following brief descriptions of normal forms will utilize examples from a relational data base; however, the concepts of normal forms are equally applicable to other types of data base systems. The discussion of the first three types of normal forms is taken from Kroenke (Ref 5).

A relation in first normal form is a relation in which attributes are single entry strings or numbers. First normal form prohibits the inclusicn of repeating groups. Repeating groups must be contained in nest relations.

Kroenke (Ref 5) gives the following example of a relation in first normal form:

STU-CLASS( [STUDENTNUM],SNAME,MAJOR,[CLASSNAME],TIME,ROOM)

The disadvantage of relations in no higher than first normal form is that information about two or more entities is stored in one relation and information can be lost. If, for example, all students taking a particular course, drop the course, or if no students are enrolled in the course, information about that course will be lost. Similarly, if a student drops all classes, information about the student will be lost. Relations of this type can also cause unexpected problems because of the relative complexity of updating requirements.

The difficulty caused by first normal form is based on the

concept of functional dependencies. The relation has a composite key, STUDENTNUM and CLASSNAME. Some of the attributes in the relation depend on part of the key, and some attributes depend on another part of the key.

That difficulty can be overcome by dividing the relation into three separate relations, each in second normal form:

STUDENT( [STUDENTNUM],SNAME, MAJOR)

CLASS( [CLASSNAME],TIME,ROOM)

STU-CLASS( [STUDENTNUM,CLASSNAME])

In second normal form, all dependent attributes depend on the entire key, not just part of it. Second normal form overcomes the difficulties caused by first normal form, but with second normal form some potential difficulties remain.

Using another example from Kroenke:

MAJOR( [STUDENTNUM],MAJOR-DEPT,DEPT-HEAD)

is a relation that is in second normal form, but not in third normal form. Deletion of a tuple from this relation can conceivably lose the name of the head of a particular department. The difficulty lies in the fact that within the dependent attributes, there is a dependency (transitive dependency) that does not depend on the key; ie, the key can determine DEPT-HEAD, but so can MAJOR-DEPT.

The difficulties caused by second normal form can be overcome by decomposing the MAJOR relation into two separate relations that are third normal form:

STU-MAJOR( [STUDENTNUM],MAJOR-DEPT)

HEAD( [MAJOR-DEPT],DEPT-HEAD)

A relation in third normal form is a relation that is already in second normal form, plus it has no transitive dependencies.

The concepts of first, second, and third normal forms were identified early in the development of the theories on relational data base systems. More recently (1976) an additional normal form, fourth normal form, has been identified (Ref 1). Each of the second, third, and fourth normal forms serve to prevent anomalies that can occur with its predecessor.

Anomalies can occur in third normal form when the dependent part of a relation consists of 2 or more attributes that are multivalued and mutually independent.

Date's example is:

CTX( [COURSE,TEACHER,TEXT])

where both TEACHER and COURSE are multivalued and mutually independent. It is in third normal form but can lead to tuples like:

| COURSE | TEACHER | TEXT |
|---|---|---|
| Physics | Prof Green | Principles of Optics |
| Physics | Prof Green | Basic Mechanics |
| Physics | Prof Brown | Principles of Optics |
| Physics | Prof Brown | Basic Mechanics |

This leads to redundancy and the use of extra storage. If another professor replaces Prof Brown, two tuples must be altered vice one. The same holds true with other changes that might be made with the data. The problems associated with that type of relation in third normal form can be overcome by using two relations that are in fourth normal form:

CT( [COURSE,TEACHER])

CX( [COURSE,TEXT])

Each relation would have two tuples rather than the four above:

CT

| COURSE | TEACHER |
|---|---|
| Physics | Prof Green |
| Physics | Prof Brown |

CX

| COURSE | TEXT |
|---|---|
| Physics | Principles of Optics |
| Physics | Basic Mechanics |

Date's definition of fourth normal form from reference 5 is: "A normalized relation R is said to be in fourth normal form (4NF) if and only if, whenever there exists a multivalued dependency in R, say of attribute B on attribute A, then all attributes of R are also functionally dependent on A".

While it may not be necessary for a designer to have all relations in fourth normal form, the designer must be aware of the potential anomalies that can arise with lesser normal forms.

F. EXTERNAL MODELS

In most cases of data base applications, user groups do not "see" or use the entire data base. Generally, user groups are limited to only that portion of the data base required for their applications.

Terms to describe that portion of the data base seen or used by a user group, and terms to describe the entire logical data base differ widely. Date (Ref 1) uses the term DATA MODEL to describe that portion of the data base used by any particular user group.

For reasons of security, a logical data base designer must be aware of the requirement for prohibiting some user groups from accessing certain information used by other user groups.

The implementations of security measures differ between data base management systems. Generally, a system of locks and passwords can be used to control access to guarded information. The

validity of the locks depend upon how the data base management system prevents unauthorized access to data. This is system dependent; however, it is considerably easier for the system software to prevent access to particular relations than to grant access to part of the relation and deny access to other parts of the relation.

Hence, the logical data base designer should insure that attributes requiring security are not assigned to relations along with attributes used by other user groups.

# IV. LOGICAL DATA BASE DESIGN

## A. GENERAL

Armed with the basic concepts just outlined, the designer of a logical data base must analyze the organization's data requirements and design a logical data base that best suits the needs of the organization.

The designer must seek an "efficient" design. Efficiency may be difficult to specify. There may be the classical time versus storage space trade-offs. There may be some retrieval requirements that must be measured in terms of seconds, while other requirements may all be predetermined, or there may be a preponderance of unique or nearly unique queries. It will more likely be the case that the organization will have a mixture of requirements.

Generally, efficiency will translate into quick retrieval times for high probability retrievals, and ease of use for programmers and casual users. These two requirements, in turn, generally translate into accessing the fewest number of relations to obtain all the information required.

Another important aspect of efficiency is the relevance of retrieved data to the requester's requirements. This in turn relates to whether there is too little or too much data retrieved and whether the retrieved data is meaningful.

A data base will most likely be dynamic - it will tend to change form and content with the passage of time. It will change because of user acceptance and the perception of potentially new applications. Hence, a major efficiency factor will be clarity

of structure as perceived by the user community. The potential anomalies and update problems discussed in conjunction with normal forms should be carefully considered.

The logical data base design process will be examined using a hypothetical data base generally built around the requirements of three major file management systems in use at the Naval Postgraduate School. The file systems are used for storing and retrieving data on students, courses, and faculty. The preponderance of the retrieval requirements are for fixed-format, periodic reports. See Appendix B for a discussion of the requirements of the data base.

Evaluation of various possible logical designs will be accomplished by using a relational data base management system call INGRES. See Appendix C.

After completion of Phase 1 of the data base design process, the designer will have identified the organization's requirements for entities and the entities' data items. Then begins the difficult task of formulating a logical data base design.

There are two phases to the logical design. One, it is necessary to identify the required relationships between entities in the data base. Two, for each entity in the data base, the entities' attributes must be grouped into relations. The attributes must include those foreign keys determined to be necessary as a result of phase one.

Guidance for this process gleened from the literature is at best vague and at worst, contradictory. Various logical

data base design techniques will be examined using the hypothetical NPS data base for analysis. Analytical techniques will also be examined. Finally, the author will present a comprehensive procedure for the logical data base design task.

## B. HEURISTIC LOGICAL DESIGN METHODS

### 1. Logical Design Based on Characteristic of Data

Kroenke (Ref 5) proposes an approach to logical design that is primarily heuristic and is based on the characteristics of the data to be stored.

The importance of a data dictionary is stressed. A data dictionary is a file containing one entry for each data item used by the organization. An extensive data dictionary could specify the item's format (conflicting formats having been resolved), where or how it is used, its frequency of use, and the files in which it is contained. The data dictionary provides the logical designer with a complete, ready reference of the information that the organization uses.

Kroenke's overall approach to the logical design is as follows: "Once the data dictionary has been constructed, it can be used to formulate a preliminary database design. This formulation is a two-phased process. In the first phase, data is allocated or grouped according to content, frequency of use, and (in some cases) size. The result is one or more database files (or records, or segments). The second phase is to complete the design by detailing which fields will be keys, how file relationships will be represented, and so on. Unfortunately,

these details depend to a large extent on the database system used. This means that the second phase must be repeated for each system under consideration."

The parenthetical reference to possibly basing decisions on the size of attributes is based on his statement that: "A 4-byte seldom used data item grouped with frequently used data is not as inefficient as a 4000-byte seldom-used item grouped inappropriately."

Before proceeding with Kroenke's approach, it is necessary to note that his guidance is for data bases in general, not specifically relational data bases. Therefore, such additional factors as normal forms and repeating groups must be considered.

The author believes that Kroenke'e guidance is over simplified in two important aspects. First, given two organizational entities that do not have any present or conceivable relationship to each other, there is little need to have them in the same data base. Hence, some thought as to how the organization's entities are inter-related should preceed the mechanics of logical design.

A second point is that an organization's use of certain data items does not necessarily imply that all those data items need to appear in the data dictionary or in the subsequent logical design. The values of some data items may be derived from the values of other data items, thus introducing some unnecessary redundancy. Additionally, it may be necessary to invent additional data items for use by the DBMS. The construction of a data dictionary should not proceed until these matters have

been resolved; otherwise the data dictionary may contain unnecessary entries and omit some necessary entries.

2. Logical Design Based on Objectives

Weiderhold (Ref 4) discusses the logical data base design process at some length. Though a rigorous approach is not presented, he outlines a number of objectives to be considered in constructing the "joint database" (DATA MODEL in Date's terminology).

The five objectives are:

(1) Construct relations with the greatest degree of semantic clarity.

(2) Construct the database using the smallest number of relations.

(3) Construct the database so that it will have the smallest number of tuples.

(4) Construct the database so that the number of data elements stored will be minimal.

(5) Construct the database so that connections between relations and shared attributes will be minimal.

There is a certain amount of mutual contradiction between objectives 2, 3 and 4 and between 3 and the concept of normal forms. The example used to illustrate conversion from third to fourth normal form shows the contradictions. In third normal form there was one relation with four tuples and each of the teacher and text attribute values appeared twice. In fourth normal form, there were two relations with two tuples each and each teacher and text attribute value appeared only once.

No doubt Weiderhold does not mean that all objectives must be met concurrently, but that each must be weighed accordingly. Weighing the importance of the contradictory objectives, however, does imply the need for a certain amount of trial and error.

## C. APPLICATION OF HEURISTIC METHODS OF LOGICAL DESIGN

The guidance provided by Kroenke and Weiderhold were applied to the task of logical data base design for the hypothetical NPS data base. The guidance was amended based on the author's previous comments on perceived deficiencies.

The first step in the logical design process should be to examine the organization's information requirements to determine how the entity interfaces should be managed. In the case of the NPS data base, there are three entities: students, courses, and faculty. The relationships between them are all "many to many". That is, students have many courses and many professors; courses have many students and many professors; and professors have many students and many courses.

A particular course can be uniquely identified by its name (40 characters) or by its number (6 characters : CS 3112). It is prudent to select the smallest format for use as a key or a foreign key. To uniquely identify a particular course meeting for students or professors, it is necessary to use a composite key, the segment number (1 byte integer) and the course number.

Students can be uniquely identified by name (27 characters), by social security number (9 characters), or by student mail

center number (SMC) (2 byte integer). For students, the student mail center code (SMC) would be a good choice for use as a key or a foreign key.

Faculty members can be uniquely identified by name (27 characters) or by social security number (9 characters). The SSN should be used.

Given n entities with the complete many to many relationships just described, it would be wasteful of space and would complicate modifications to the data base if all n entities incorporated the n-1 foreign keys in its relations to form the links with other entities. Known or anticipated requirements of the data base should be examined to determine the best way to form the required links, some of them indirectly.

Each foreign key embedded in a relation belonging to another entity carries with it the cost of additional storage. The minimum storage cost is incurred when each entity can directly link to only one other entity. Given the three relations

STU-COURSE([STUDENT,COURSE])

COURSE-PROF([COURSE,PROF-NAME],COURSE-INFO)

PROF([PROF-NAME],PROF-INFO)

each contains only one foreign key. Thus, given a student, in order to find information on one of the student's professors, it is first necessary to use COURSE to link to COURSE-PROF, and then use PROF-NAME to link to PROF. It would be possible to determine the professor information directly if PROF-NAME was included as a foreign key in STU-COURSE. That design yields

a faster retrieval but requires more storage. It is also only in second normal form, with the dangers that entails.

With the design of those three relations, it is a simple matter to link from STU-COURSE to COURSE-PROF and then to PROF using the foreign keys. It is not necessarily a simple matter to link in the other direction. To start with PROF and link to COURSE-PROF, it is necessary to search all tuples of COURSE-PROF until a match is found for the desired value of PROF-NAME. Hence, a link in one direction is not the same as a link in the other direction. In the example given, if there were n different tuples in COURSE-PROF, one would have to expect n / 2 comparisons, on the average, to obtain the desired match. If the professor had an unknown number of courses and all were desired, then all n tuples would have to be searched.

This difficulty can be overcome in one of two ways. Embedding a COURSE foreign key in the PROF relation would allow a direct link. Alternatively, structuring COURSE-PROF with an inverted file on PROF-NAME would give essentially the same result. Both methods, however, require additional storage space.

The designer of the logical data base must identify all probable binary associations among the organization's entities. Trade-offs between storage space and allowable retrieval times must be made. Those binary associations that occur frequently, or have a high probability of occurrence, must be accommodated by embedding the appropriate foreign keys.

For the NPS data base several links are known to be required.

A student's past courses and grades are required. A student's present course and segment numbers and professors are required. Class rosters showing all students in that course and segment are required. There is no fixed requirement to list all students belonging to a professor.

A student relation in fourth normal form that would contain attributes on courses and professors would have approximately 3000 tuples (approximately 1000 students averaging 3 courses apiece). A foreign key for course is mandatory. The requirement to retrieve the name of the professor for that course also exists, but that foreign key would cost 9 x 3000 = 27000 bytes. The number of courses and the number of professors are both considerably smaller than the number of students. Since the retrieval times are in terms of days, rather than the 27000 byte foreign key for professor, the "n / 2" search to get the professor for a student's course is probably the most efficient configuration.

Hence the best foreign key arrangement would be as follows. For a student's past courses, the course number included in a student relation will suffice. For a student's present courses, the course number and segment number will enable the data base system to make the required links.

The requirement for class rosters can be met by including students' SMC numbers in a course relation. Linking professors with course-segments can be accomplished through using course and segment numbers as a foreign key within a professor relation. Identifying all students of a professor can be accomplished

indirectly through the class rosters of the professor's course-segments. Linking students with their professors can be through a match on course and segment numbers.

Having identified how entity interfaces will be managed, the next step is to analyze the organizations required data items. The purpose of the analysis is to firmly determine the attributes that must be included in the data base. The analysis must accomplish two goals: (1) delete from consideration those data items that can be derived or implied from other data items; and (2) identify additional attributes whose requirement is implied by the data items.

For the NPS data base, items can be eliminated. First, there is a requirement for both a student's SSN and country. The presence of a valid 9 digit SSN, however, implies that the student's country is the United States. A foreign student will not have a valid SSN, so those 9 bytes can be used for the student's country, or abbreviation thereof. Second, there is a requirement for a student's past grades and also the student's credit points for a given course. Given the course's credit hours and the student's grade, the credit points can be determined. Similarly, the grade can be determined from the hours and points. Hence, either the grade or the points is needed, not both.

There is a requirement to identify two relatively small groups of students: section leaders and woman students. That identification can be accomplished by giving all students 2 attributes of 1 byte each to indicate whether or not they belong

to those two groups. A second approach would be to form two relations each containing identification of students in those two groups. A third and better approach would be to see if membership in those two groups of students can be implied by the value of some other data item. That is in fact the case. The SMC numbers range in value from 1000 - 3015 requiring a 2-byte integer. The 2-byte interger, however, can store values far in excess of 3015. A male section leader can be identified by adding 10000 to his SMC. A woman student can be identified by adding 20000 to her SMC and a woman section leader 30000. In all cases, the valid SMC is the stored SMC modulo 10000.

The second step in the analysis of the data items is to determine if the use of the organization's data implies the requirement for any additional data items. That is, in fact, the case with the NPS data base. Storing information on a student's previous courses at NPS implies the requirement to know when the student took the courses. That implies the need for an additional data item. That data item would be an interger value giving the student's quarter in which he or she took the courses. Retention in the data base of a student's future course requirements also implies the requirement to identify the quarter in which the student needs the course. That results in a data item to identify the academic year and quarter in which the course is needed. That can be done with a three character item such as 782, which would be the second quarter of academic year 1978.

That completes the analysis of the organization's data items.

They can now be considered attributes to be included in the data base.

Based on certain reasonable assumptions about numbers of students, courses and professors, the frequency of use of data items for the NPS file systems was determined per academic quarter. Representative frequencies are as follows: (For known, current, periodic reports)

| | |
|---|---|
| Student name | 59150 |
| Student rank | 53650 |
| Student address | 8000 |
| Student previous degree | 3000 |
| Course number | 61875 |

While determining frequency of use is straightforward, grouping by content is not; it is subject to interpretation. The author's interpretation is that of grouping data attributes that tend to serve a common purpose.

In analyzing student attributes, there appear to be 5 content groupings: (1) students' identifying information; (2) students' professional, military-related information; (3) students' local information; (4) students' NPS educational information; and (5) students' prior educational information.

Similarly, courses have (1) identifying information; (2) class rosters; and (3) spatial (time and place) information. Professors have (1) identifying information and (2) course information.

Analysis of frequency of use, content, size and use of the

concept of normal forms leads to the following logical data base design:

STU-MOST([NAME],SMC,RANK, SERVICE, CURRICNUM,DESIG,SSN)

STU-PRO([SMC],DOR,LINEAL,YEARGROUP,PRD,DATEREPORT)

STU-LOCAL([SMC],ADDRESS,CITY,PHONE)

STU-ED([SMC],SECTION,STARTDATE,GRADDATE,DEGREE-EXPECT)

STU-PRIOR-ED([SMC],PDEGREE,SCHOOL,YEAR)

STU-NPS-PAST([SMC]),STU-QTR,COURSENUM,PTS)

TOT-QPR([SMC],TOT-ATTEMPT,TOT-PASS,TOT-PTS,TOT-QPR,QTR-TOT-QPR,TIMESDL)

GRAD-QPR([SMC],GRAD-ATTEMPT,GRAD-PASS,GRAD-PTS,GRAD-QPR, ATR-GRAD-QPR)

STU-PRESENT([SMC,COURSENUM,SEGNUM],PTS)

STU-REQUIRE([SMC,YR-QTR,COURSENUM])

STU-THESIS([SMC],TITLE,ADVISOR)

COURSE-ID([COURSENUM],COURSENAME,HRS)

ROSTER([COURSENUM,SEGNUM,SMC])

COURSE-LECT([COURSENUM,SEGNUM],MON-HR,MON-LOC,TUE-HR,TUE-LOC, WED-HR,WED-LOC,THR-HR,THR-LOC,FRI-HR,FRI-LOC)

COURSE-LAB([COURSENUM,SEGNUM],DAY,START-HR,STOP-HR,LOC)

CURRIC([CURRICNUM],CURRIC-NAME,COFF-CODE,COFF-NAME)

PROF-PERS([PNAME],PSSN,DEPT,MAIL-CODE)

PROF-COURSE([PSSN,COURSENUM,SEGNUM])

See Appendix D for a data dictionary giving an explanation of the attributes used above.

Using Kroenke's approach, a logical data base design has

been developed. All relations are in fourth normal form except CURRIC and PROFPERS. (CURRIC is in second normal form and PROFPERS is in third normal form assuming more than one department is allowed. CURRIC can remain in second normal form since there is little chance of losing curricular office information because of having no students in a particular curriculum.)

Little can be said about the "optimality" of the design, except that it is probably not optimal. While frequency of use can be determined with a good degree of accuracy, the actual use of that data is intuitive. Having identified the frequency of use, where do the "dividing lines" go? SECTION is used an estimated 24000 times per quarter (24 times per student) and PRD (Projected Rotation Date) 29000 times. Should they be grouped together? Probably not; the intersection of their usage is small.

The concept of "content" is very imprecise. Given the original list of student attributes, grouping by content can yield a number of different arbitrary groupings of attributes.

The deficiencies in the reliance on frequency of use and content stem from its lack of theoretical foundation. Such a foundation could be built upon the basic concepts discussed in Section III.

The methods explored thus far leave many intuitive decisions up to the designer. Something is needed to reduce the number of arbitrary design decisions that must be made. A mathematical model could be useful in aiding the designer with many of the

decisions regarding the allocation of attributes to relations. Even if a mathematical model could not identify "the" optimal solution, if properly formulated the model could enhance the designer's chances of developing a good design.

## D. ANALYTICAL TOOLS FOR LOGICAL DESIGN

### 1. System and Application Considerations

In order to formulate a mathematical model, precise quantities related to the data and the data base usage must be identified. From an analysis of the way that the organization uses the data, the designer will have identified each attribute's frequency of use. That is a precise mathematical quantity that definitely must influence logical design decisions.

There will be certain hardware-dependent and DBMS-dependent factors that can be identified. These factors could be related to storage or I/O operations in the system. For the system used by the author, the line printer has 77 print positions. If text exceeding 77 characters in length is sent to the line printer, characters past the 77th are lost. The interactive use of the system is through CRT terminals. The CRT terminals will accept as input 72 characters without wrap-around and 79 characters with wrap-around.

Hardware features such as these can be important considerations in logical data base design. Auditing of a data base may require periodic print-outs of the contents. If a relation exceeded 77 print positions, some data would be lost. Data entry to the data base would probably be by cards (80 positions maximum) or

CRT terminal (79 positions maximum).

The number of print positions required by attributes in the INGRES system are not necessarily the formats used for storage of the attributes. (See FORMATS, Appendix B)

The frequency of use, formats, or print position parameters could be of use if a proper type of mathematical model could be identified.

2. The Interger Programming Problem

The problem facing the logical designer is somewhat like the classical "backpacking" problem, often solved by implicit enumeration of a (0/1) integer programming problem (Ref 8). In the backpacking problem, the task is to analyze a known number of quantities and assign them to a receptacle. The quantities will not all fit into the receptacle. The problem-solver must assign relative values to the quantities and assign relative costs that would be incurred if a given quantity is chosen. The values are represented in an objective function to be maximized or minimized, and the costs or restrictions are represented in one or more constraints.

Gillette's formulation of a representative backpacking problem (Ref 6) is as follows:

Maximize $Z = 6 X1 + 3 X2 + X3 + 5 X4$

Subject to: $5 X1 + 2 X2 + X3 + 3 X4 <= 5$

$all\ X = 0\ or\ 1$

The coefficients in the objective function specify relative values of each item, and the coefficients in the constraint

represent costs or limitations on each item; in this case the constraint deals with the weight of each item, and the total weight of all items selected cannot exceed 5. If there is an additional constraint that two items could not be assigned together (ie, neither, or either, but not both), the additional constraint could take the form:

$$X1 + X3 <= 1$$

The backpacking problem deals with a known number of receptacles and quantities, but the designer must take a known number of items (attributes) and put them all into an unknown number of receptacles (relations). This difficulty can be overcome by viewing the logical design problem as a series of backpacking problems. Subject to certain values and costs, the designer can assign some attributes to a relation, delete those attributes from further consideration, and repeat the assignment process. The (0/1) integer programming problem can be repeated until no attributes remain unassigned.

Before attempting to formulate an integer program, some consideration must be given to the attributes that must appear in the problem. Forty-five student attributes have been identified in the hypothetical NPS data base. Some of them appear in repeating groups. Those attributes to be used in repeating groups must be assigned to relations based on the requirements of fourth normal form. The formation of relations for repeating groups is straight forward. Those attributes need not be considered in the (0/1) integer programming problem.

In considering an organization's entities, there may be some attributes for which values must be assigned for only a relatively small percentage of the occurrences of the entities, ie, possibly not every occurrence of a given entity will have an occurrence of a particular attribute. That is the case with the NPS entities. Not every student will have a thesis title and thesis advisor. It would be wasteful of space to reserve storage for thesis title and advisor for all students if only 30% will have values assigned to those attributes. Hence, attributes of that nature should be used in their own relations, with an overall net savings of storage and time.

There remains 32 student attributes to be considered. Solving an interger programming assignment problem with 32 variables is a considerable task. If there is some reasonably accurate way to reduce that number, then formulation of the problem could be simplified.

In the NPS data base, there are certain clusters of attributes that will always, or almost always, be retrieved together. One of the goals of the design is to simplify retrievals by judicious placement of the attributes. It is reasonable, then, that attributes that have a very high probability of being retrieved together, should be together in the same relation. Hence, if certain clusters are considered to be "attribute groups" prior to the assignment, the value of the result of the integer programming problem is not diminished.

For the student entity in the NPS data base, address, city

and telephone number are almost always retrieved together. Similarly, prior school and prior degree and year, and all of the quality point rating attributes are all retrieved as clusters. Consolidating these attributes into groups reduces the 32 attributes to a more manageable 18.

Formulating an integer programming assignment problem requires an objective function to be maximized (or minimized) plus constraints. The variables throughout the objective function and the constraints represent the attributes. The coefficients in the objective function must represent some sort of weight or value assigned to the attribute group. The coefficients in the constraints represent some sort of cost or limitation.

For the objective function, a meaningful weight or value must be identified for the attribute groups. For the hypothetical data base, the frequency of use of each attribute (group) is known. Also known is the number of reports in which they appear. Since 60 of the reports are quarterly (1 is semi-annual), there is little lost if the number of reports in which the attribute (group) appears is used as a surrogate for the frequency of use. Hence, the coefficients in the objective function will be the number of reports in which the attribute (group) appears. A data base must be able to respond to unusual queries. In the absence of specific information indicating the probable nature of these queries, it must be assumed that either they will tend to use the same attributes as the known requirements do, or they will have to accept longer response times.

The inequalities in the constraints must express the costs or limitations imposed on the attribute groups. One of the costs associated with an attribute (group) is the number of bytes of storage; another is the number of print positions required to display it. There are limitations on the assignment to relations in terms of external models - for privacy or other security reasons, attributes used only by one user group may have to be prohibited from being mixed with attributes used by other user groups. These types of limitations can and should be expressed in the form of constraints. Within the NPS data base, QPR information is used only by the Registrar and must be kept separate from information accessed by other user groups.

Following are the variables used to represent the attribute (groups) in the integer programming problem:

| | |
|---|---|
| A Name | J Section |
| B Rank | K Degree Expected |
| C Curriculum Number | L Reporting Date |
| D Service | M Local Information |
| E SMC | N Professional Information |
| F Designator | O Total Quality Point Rating |
| G SSN | P Graduate Quality Point Rating |
| H Projected Rotation Date | Q Prior Education |
| I Graduation Date | R Start Date |

The complete integer programming assignment problem then becomes:

Maximize: Z = 56A + 48B + 45C + 45D + 35E + 35F + 31G + 29H +
20I + 14J + 12K + 9L + 9M + 6N + 5O + 5P + 3Q + 1R

Subject to: 27A + 6B + 6C + 6D + 6E + 6F + 9G + 6H + 6I + 6J +
6K + 6L + 34M + 18N + 56O + 50P + 48Q + 6R <= 76

O + H <= 1 (one or the other)

O + R <= 1 (one or the other)

O + N <= 1 (one or the other)

(Since the objective function is a "maximize", it must be changed to a "minimize". That is done by changing all variables from A, etc., to 1 - A', etc.) (For the actual assignment model, the constraints must be in the form of "<=".)

The above formulation of the (0/1) integer programming problem is suitable for assigning attributes to the first student relation.

Subsequent iterations are similar. Since a key must be provided for second and subsequent relations, the first constraint inequality must reflect the fact that the key must be included in the allowable print positions. Since SMC is the best key, and it requires 6 print positions, the "76" in the first constraint must be changed to "70" for subsequent iterations.

The use of the print position limitations in the constraint is an arbitrary choice for this problem; some expression of storage limitations may be more appropriate in other applications. Use of the more stringent limitation of 76 print positions in this case is more meaningful for illustrating the nature of the problem.

Repeated iterations of the integer program results in the

following logical data base design:

STU-MOST([NAME],RANK,CURRICULUM,SERVICE,SMC,DESIG,SSN,PRD)

STU-DATA([SMC],SECTION,DEGREEXPECT,REPORTDATE,ADDRESS,CITY,
PHONE,GRADDATE,STARTDATE)

STU-PRIOR([SMC],PDEGREE,SCHOOL,YEAR,DOR,LINEAL,YRGROUP)

TOT-QPR([SMC],TOT-ATTEMPT,TOT-PASS,TOT-PTS,TOT-QPR,QTR-TOT-
QPR,TIMESDL)

GRAD-QPR([SMC],GRAD-ATTEMPT,GRAD-PASS,GRAD-PTS,GRAD-QPR,QTR-
GRAD-QPR)

STU-THESIS([SMC],TITLE,ADVISOR)

STU-NPS-PAST([SMC],STU-QTR,COURSENUM,PTS)

STU-PRESENT([SMC,COURSENUM,SEGNUM],PTS)

STU-REQUIRE([SMC,YR-QTR,COURSENUM])

COURSE-ID([COURSENUM],COURSENAME,HRS)

ROSTER([COURSENUM,SEGNUM,SMC])

COURSE-LECT([COURSENUM,SEGNUM],MON-HR,MON-LOC,TUE-HR,TUE-LOC,
WED-HR,WED-LOC,THR-HR,THR-LOC,FRI-HR,FRI-LOC)

COURSE-LAB([COURSENUM,SEGNUM],DAY,START-HR,STOP-HR,LOC)

CURRIC([CURRICNUM],CURRIC-NAME,COFF-CODE,COFF-NAME)

PROF-PERS([PNAME],PSSN,DEPT,MAILCODE)

PROF-COURSE([PSSN,COURSENUM,SEGNUM])

Many of the student attributes in the hypothetical NPS data base appear in repeating groups which are not effected by the proposed integer programming assignment procedure. Hence, in both logical designs presented, many of the relations remain the same.

Single valued attributes (only one value per entity occurrence) are affected by the proposed procedure. Savings in both storage and execution time are apparent in the integer programming design.

The storage scheme in INGRES is that tuples are stored on 512-byte tracks. For each track used, there is a ]2-byte overhead, leaving 500 bytes for tuple storage. Tuples are not split between tracks. The wasted storage per track is 500 modulo the tuple length in bytes.

The design derived from the integer program requires 14 fewer tracks for 1000 students than the design derived by heuristic means.

Retrieval speed can vary widely between data base management systems. Perhaps the only generalization that can be made regarding retrieval speed is that given one retrieval requirement from two different logical designs, the design requiring the fewest relations for all the information will result in the faster retrieval.

In analyzing the 61 known reports for the hypothetical NPS data base, in no case does the integer program design require more relations to complete the retrieval than the heuristic model. For 8 of the reports, the integer program design requires 2 fewer relations and for 13 reports 1 fewer.

A timing analysis was conducted for retrieving the information required for 20 of the reports. The INGRES query language was embedded in the general purpose language "C".

Different subroutines were used for the different retrieval requirements from the two different designs. The timing mechanism was a bit crude - in a single user mode, a system function for returning wall-clock time before and after execution of the subroutines was used. Though not precise, that method was able to allow general comparisons of the relative timing for the different retrievals. Identical logic was used in all the subroutines.

In retrieving the complete required information for certain reports, the integer program design completed the queries in approximately 1/3 the time required by the heuristic design.

Certainly, the heuristic design could have been made as "bad" as possible to improve the relative appearance of the assignment model design. However, the heuristic design was the result of several attempts to diligently apply the heuristic guidance. Additional trial and error could have probably resulted in a better heuristic design. A key point, however, is that no trial and error was involved in the integer program design - the mathematics of the model made the decisions once the coefficients for the model had been chosen.

3. Analysis of Coefficients for the Integer Program

The integer programming approach is a precise method, provided that the coefficients used are indeed correct measures of the relative values and costs.

The coefficients seem to work properly in this particular case because of the nature of the attributes. It cannot

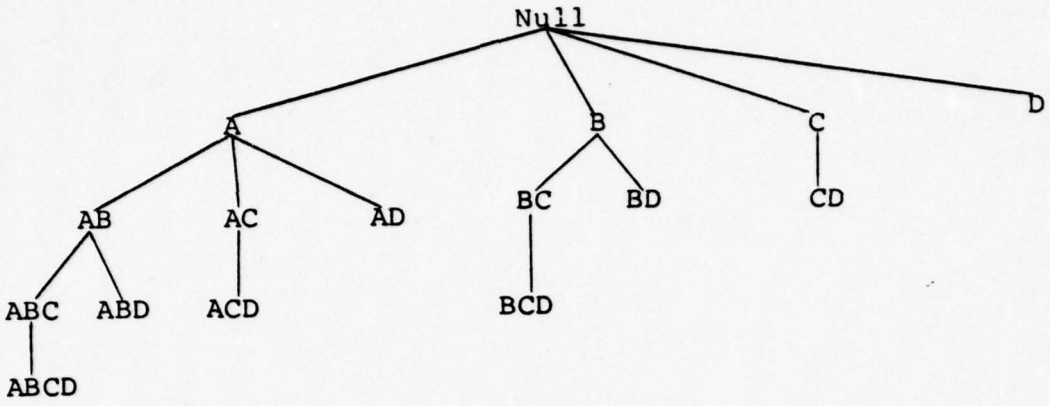
necessarily be stated, however, that in the general case the number of reports in which an attribute appears is the proper coefficient for use in the objective function.

Since the choice of coefficient may or may not be correct, a method is required to either confirm that the choice of coefficient was correct, or find a better choice.

The difficulty with strictly using the number of reports in which an attribute appears, is that the number does not measure the mutual usage between attributes that have nearly the same frequency of use. For example, in the NPS data base, PRD (projected rotation date) is used in 29 reports and SSN in 31. Since the values are very close, the integer programming model will probably assign them to the same relation. However, their intersection is only 13 (used together 13 times), so they are not as strongly bound as the numbers 29 and 31 would indicate.

Matrix methods can be used to measure the connections between attributes. A 3 dimensional matrix (attributes x attributes x reports) can be used to determine how often any two attributes are used together. For usage of three attributes together, a four dimensional matrix is required. Similarly, 8 requires a 9 dimensional matrix. That is not very useful.

A better method is available for measuring the intersections of n attributes together. A tree can be constructed to represent the power set of n attributes. A sample tree for four attributes would be as follows:




Each letter represents one attribute. Each node represents the frequency with which that combination of attributes occurs.

Using a program developed by HOLL, an analysis of the mutual frequency of attribute usage was obtained. That analysis was used to verify the correctness of the attribute groupings provided by the integer programming model. Input to the program was a two-dimensional matrix (reports x attributes). All entries in the matrix are either 0 or 1 - 1 if the attribute is required for the report, 0 otherwise.

The goal in assigning attributes to relations is to get the group that will be used together most frequently while meeting the system constraints. The initial run on the assignment model resulted in the assignment of 8 attributes.

The mutual frequency of those attributes was easily obtained from analysis of report information. The entire group of eight was used 9 times. The tree search was completed to see if any other group of attributes meeting the constraints occurred more

often than those eight. There were no such groups. Hence, the selection of those eight attributes for the first relation was correct, implying that the selection of coefficients in the integer programming objective function was adequate.

For each successive iteration of the integer programming model, a similar analysis was conducted, confirming the correctness of the assignment.

Since the tree search technique is a valid means of analyzing the mutual known usage of attributes together, it is reasonable to consider whether the technique could be used to select coefficients for the integer program model, or indeed, even identify groupings of attributes into relations directly. The tree search can do neither. The tree in this case has over 260,000 nodes in it. The model cannot be structured in a manner that would allow representation of any of the system or application-dependent constraints. The value assigned to each node in the tree is a property of all attributes represented at that node, not the property of any one attribute.

In the general case, the coefficients for the integer programming objective function can be chosen to represent the application-dependent value of each attribute. Subsequent to the assignment of attributes to a relation, the tree search can be used to confirm the assignment.

Should the tree search fail to confirm the choices of attributes from the assignment model, it will have identified a better solution, assuming the group with the higher frequency did, indeed, meet the constraints.

# V. CONCLUSIONS

The complex task of logical data base design for a relational data base can be greatly simplified by the use of operations research methods and some of the theory associated with relations and relational data bases.

While "the" optimal solution cannot be guaranteed, judicious application of the appropriate methods can achieve an efficient logical design, as efficiency is defined by the designer.

As a result of Phase I of the overall design process, the designer will have identified the organization's entities that must be represented in the data base and the data items associated with those real-world entities. The designer should then proceed as follows:

1. Identify the real-world relationships between entities that must be represented in the data base. Analyze the required links between entities, eliminating from consideration those links that will have a low probability of occurrence, or those that can more efficiently be represented through transitive links. Select the best foreign key arrangement for completing the required links between entities, considering inverted files usage where appropriate.

2. Analyze each entity's data items, including the required foreign keys from step 1. Eliminate those data items that are redundant or can be derived from other data items. Add data items that will be required by the data base management system to appropriately identify and manipulate other data items. The resulting list of data items can now be considered attributes

for inclusion in relations in the data base.

3. Identify, for each entity, those attributes that are used in repeating groups. Form those attributes into nest relations or associative relations in fourth normal form, including as a key, or portion of a key, an identifying attribute of the parent entity. The identifying attribute will be chosen so as to provide semantic clarity, efficient storage utilization, and quick identification.

4. Identify attributes that will belong to a small number of occurrences of the parent entity (like thesis information for the NPS data base, or perhaps some sort of merit pay or bonus plan for an organization). Group those attributes as in step 3.

5. If the number of remaining attributes is small, or if the remaining attributes have such dissimilar use that no clustering of attributes is possible, proceed with step 6. Otherwise, form clusters of those attributes that have similar content and a high degree of concurrent use. Those clusters form a reduced number of attribute groups.

In this context, "small" is relative. If the computer system available for solving the integer programming problem is capable of handling large problems, then "small" may in fact be rather large (say, 40).

6. Formulate a (0/1) integer programming model to successively assign attributes (or attribute groups) together into relations. The integer programming model will consist of an

objective function to be maximized or minimized and constraints. Variables in the objective function will represent the individual attributes (or attribute groups).

Coefficients in both the objective function and the constraints will be system and application dependent. Coefficients in the objective function must represent some measure of the relative value of each attribute or group. In most cases, that value will be dependent upon the frequency of use or probability of use.

Coefficients in the constraints will represent both costs and limitations. The costs will be related to either storage or I/O restrictions. Limitations will be related to the concept of external models, that is, prohibitions against one attribute being stored with others.

## APPENDIX A

## THE RELATIONAL DATA BASE CONCEPT

Relational data bases are founded upon the mathematical theory of relations. A relational data base is composed of a series of relations. Relations are two-dimensional tables such as the three shown below (from Ref 1):

SUPP

| S# | SNAME | STATUS | CITY |
|---|---|---|---|
| S1 | Smith | 20 | London |
| S2 | Jones | 10 | Paris |
| S3 | Blake | 30 | Paris |
| S4 | Clark | 20 | London |
| S5 | Adams | 30 | Athens |

PART

| P# | PNAME | COLOR | WEIGHT |
|---|---|---|---|
| P1 | Nut | Red | 12 |
| P2 | Bolt | Green | 17 |
| P3 | Screw | Blue | 17 |
| P4 | Screw | Red | 14 |
| P5 | Cam | Blue | 12 |
| P6 | Cog | Red | 19 |

SP

| S# | P# | QTY |
|---|---|---|
| S1 | P1 | 3 |
| S1 | P2 | 2 |
| S1 | P3 | 4 |
| S1 | P4 | 2 |
| S1 | P5 | 1 |
| S1 | P6 | 1 |
| S2 | P1 | 3 |
| S2 | P2 | 4 |
| S3 | P3 | 4 |
| S3 | P5 | 2 |
| S4 | P2 | 2 |
| S4 | P4 | 3 |
| S4 | P5 | 5 |
| S5 | P5 | 4 |

The names of the colums in the relations are attributes. P# (part number) is an attribute. The range of values that an attribute can assume is called a domain. All values appearing in a column must be homogeneous; that is they must all come from the same domain.

Rows in a relation are tuples, or more precisely, n-tuples for a relation having n columns. A row in a relation having 2 attributes (columns) is a 2-tuple, with 5, a 5-tuple. All rows in a relation must be unique; a tuple must differ from each

other tuple in the relation in the value of at least one attribute.

Rows in a relation are uniquely identified by a key. In the SUPP relation, S# is the key. In the PART relation, P# is the key. In the SP relation, neither S# nor P# alone can uniquely identify the row, so the key is the combination of S# and P#.

The benefits derived from integrated data bases in general certainly accrue to relational data bases. Whereas plex structures and many-to-many relationships in other data base systems require modifications to the natural structures of the data into tree structures, relational systems can represent data and relationships as they exist.

One of the chief benefits of data base systems is the capability to "link" entities together to portray their relationships. Links between relations belonging to different entities are formed by finding identical values for attributes that the relations have in common. In the sample data base, for instance, using the value of part number P1, it is possible to find the weight of all P1's supplied by supplier number 1. Similarly, it is possible to determine the number of red parts supplied from London, by using links between all three relations.

In the sample data base, relations SUPP and PART are entity relations, and SP is an associative relation providing information about an association between two entities. (See RELATIONS in BASIC CONSIDERATIONS)

Relational data bases can be portrayed in tabular form, as

above, or in a narrative form that lists the relation name followed by its attributes in parenthesis. Throughout this thesis, keys will be identified by square brackets [ ]. In a narrative form, the three relations above are:

SUPP([S#],SNAME,STATUS,CITY)

SP([S#,P#],QTY)

PART([P#],PNAME,COLOR,WEIGHT)

## APPENDIX B

## REQUIREMENTS FOR NAVAL POSTGRADUATE SCHOOL DATA BASE

The Registrar maintains an extensive file system for maintaining past and present academic information on all students, plus a limited amount of personal and professional information. Twenty-four different reports are generated using the information in the file system. A few examples are: class rosters, student grade reports, lists of graduates, etc. All reports are generated on a quarterly basis. Response time is measured in days.

The school's ADP Officer maintains a file system for personal and professional information on students. The file system is used to generate reports of an administrative nature. There are thirty three different types of reports generated by the file system. Examples of lists of student by curriculum, lists of students by rank and service, and lists of foreign students by country. These reports are generated on a quarterly basis and response time is measured in terms of days.

The Programs Officer maintains a system for quarterly scheduling of courses. Every second quarter, student demand for courses for the subsequent four quarters is analyzed. That analysis is provided to department chairmen who return to the Programs Officer a list of what courses will be scheduled in the subsequent four quarters. On a quarterly basis, a list of courses for that quarter is published. Also on a quarterly basis, the Programs Officer publishes lists showing the courses for which groups of students are registered for the next quarter.

A scheduler within the Academic Administration Department uses the list of courses for a quarter to make meeting time and room assignments.

Phase I of the data base design process was completed. The content, frequency, and response time for all reports were analyzed. Data item requirements of the organization were identified.

Analysis of organization requirements identified three entities about which data items must be maintained: students, courses, and faculty. These entities require data items as follows:

STUDENTS

Name
Rank
Social Security Number
Service
Designator (Military skills identifiers)
Country
Courses previously taken (with credit-hours, segment numbers and grades)
Courses being taken now (with credit-hours and Professor)
Graduate and total credits both attempted and passed
Graduate and total quality points earned at NPS
Graduate and total quality points earned the previous quarter
Number of times on the Dean's List
Section (group of students in same curriculum starting the same time)
Curriculum
Date reported
Date started instruction
Estimated graduation date
Projected rotation date
Entrance credits (prior school, degree, year)
SMC (Student mail center number)
Thesis title
Thesis advisor
Degree expected
Local address
Local city

Local phone number
Date of rank
Lineal number in rank
Year group
Curricular Office
Future course number requirements
Identification as a section leader
Identification as a woman student

COURSES

Course number
Course name
Course sequence number
Credit hours
Student demand for academic year
Students enrolled
Professors
Meeting time and location for lecture
Meeting time and location for laboratory

FACULTY

Name
Social security number
Department
Courses taught
Course segments taught

# APPENDIX C

## INTERACTIVE GRAPHICS RETRIEVAL SYSTEM (INGRES)

### I. GENERAL

Analysis of logical data base design methods explored in this thesis have been evaluated using the Interactive Graphics Retrieval System (INGRES) developed by the Department of Electrical Engineering and Computer Science at the University of California, Berkeley.

This appendix provides a brief description of INGRES and its query language QUEL. More detailed explanations of the features, design and implementation of INGRES are contained in Ref 6.

INGRES is implemented in the UNIX operating system developed by Bell Laboratories. At NPS the system operates on a Digital Equipment Corporation PDP-11/50.

INGRES is a relational data base management system employing the concept of logically storing attributes in relations. (See Appendix A)

Interaction between a user or applications of programs and tje data base is managed through a query language called QUEL. QUEL is based on relational calculus (Ref 7) and bears some resemblance to Codd's data sublanguage called ALPHA.

### II. QUEL - THE INGRES QUERY LANGUAGE

QUEL uses the concept of a "tuple variable". The tuple variable is a user-selected abbreviation for a particular relation in the data base. A tuple variable is defined by a

range statement like

```
RANGE OF P IS PARTS
```

where PARTS is a relation in the data base. Subsequent to the range statement, P is synonomous with PARTS. The choice of the letter as the tuple variable is arbitrary. The same letter cannot be used as a tuple variable for two relations.

QUEL provides commands for the user to CREATE relations, DESTROY relations, APPEND (add) a tuple to a relation, REPLACE (update) a tuple and DELETE an entire tuple.

Commands requiring manipulation of a tuple, or a series of tuples, require additional information that identifies the desired tuples. That additional information is referred to as the "qualification".

The qualification provides the data base management system with the means by which individual tuples can be identified. Where two or more relations are involved in a command, the qualification also provides the data base management system with the identification of the common attributes that serve as links between the relations.

Using the three sample relations in Appendix A, the following QUEL commands would determine the cities from which part number one was supplied.

```
RANGE OF S IS SUPP
RANGE OF A IS SP
RETRIEVE (A.P#,S.CITY)
WHERE A.P#="P1"
```

(Using the tuple variables, A.P# is the attribute P# in the relation SP, and S.CITY is the attribute CITY in the SUPP relation.)

The query will return two tuples:

```
P1    LONDON
P1    PARIS
```

## III. FORMATS

Attributes in INGRES can be formatted as character strings (1 to 255), integer (I1, I2, or I4) and floating point (F4 or F8).

The output formatting of INGRES (either at an interactive terminal or line printer) merits consideration. No attribute is printed with fewer than 6 print positions. Storage formats I1, I2, and C 1 - 6 require 6 print positions. I4 format requires 14 print positions and F4 and F8 require 10 print positions. Character formats in excess of 6 use the expected number of print positions.

The output conventions should be considered by anyone implementing a data base in INGRES or any similar DBMS. A 10-tuple composed solely of attributes in I4 format would require only 40 bytes of storage, but would require 120 print positions to display.

## APPENDIX D

### DATA DICTIONARY FOR DATA BASE

| ATTRIBUTE | FORMAT | REMARKS |
|---|---|---|
| ADDRESS | c21 | Local address |
| ADVISOR | c27 | Name of thesis advisor |
| CITY | c2 | Abbreviation of local city |
| COFF-CODE | i2 | Curricular office code |
| COFF-NAME | c36 | Curricular office name |
| COURSENAME | c40 | Course name from catalog |
| COURSENUM | c6 | Course number from catalog |
| CURRIC-NAME | c31 | Curric name per catalog |
| CURRICNUM | i2 | Curriculm number from catalog |
| DATEREPORT | c6 | Date reported day month year |
| DAY | i1 | Day of lab |
| DEGREE-EXPECT | c2 | Degree expected from NPS |
| DEPT | c2 | Academic department |
| DESIG | i2 | Military designator or MOS |
| DOR | c6 | Date of rank |
| FRI-HR | i1 | Lecture hour Fri 1 is 0800 |
| FRI-LOC | c5 | Location for Fri class 2 bldg 3 room |
| GRAD-ATTEMPT | f4 | Grad credits attempted |
| GRADDATE | c6 | Grad date day month year |
| GRAD-QPR | f4 | Grad quality point rating |
| GRAD-PASS | f4 | Grad credits passed |
| GRAD-PTS | f4 | Grad quality points |
| HRS | i2 | Credit hours 10 times lec + lab |
| LINEAL | 12 | Lineal number within rank |
| LOC | c5 | Location of lab |
| MAIL-CODE | c4 | Professors' mail code |
| MON-HR | i1 | Lecture hour Mon 1 is 0800 |
| MON-LOC | c5 | Location for Mon class 2 bldg 3 room |
| NAME | c27 | Name of student |
| PDEGREE | c2 | Abbreviation for degree |
| PHONE | c7 | Local phone number |
| PNAME | c27 | Professor name |
| PRD | c6 | Projected rotation date |
| PSSN | c9 | SSN of professor |
| PTS | f4 | Credit points |
| QTR-GRAD-QPR | f4 | Last quarter grad qpr |
| QTR-TOT-QPR | f4 | Last quarter total qpr |
| RANK | c5 | Military rank |
| SCHOOL | c30 | Previous school |
| SECTION | c4 | Student sections |
| SEGNUM | i1 | Course segment number |
| SERVICE | c4 | Branch of military service abbreviated |

| ATTRIBUTES | FORMAT | REMARKS |
|---|---|---|
| SMC | i2 | Student mail center number |
| SSN | c9 | SSN of student |
| SSN | c9 | Country for foreign students |
| STARTDATE | c6 | Date started instruction day mon year |
| START-HR | i1 | Starting time of lab |
| STOP-HR | i1 | End of lab |
| STU-QTR | i2 | Student's quarter for prior courses |
| THR-HR | i1 | Lecture hour Thur 1 is 0800 |
| THR-LOC | c5 | Location for Thur class 2 bldg 3 room |
| TIMESDL | i1 | Time on Dean's List |
| TITLE | c56 | Title of thesis |
| TOT-ATTEMPT | f4 | Total credits attempted |
| TOT-PASS | f4 | Grad credits passed |
| TOT-PTS | f4 | Total quality points |
| TOT-QPR | f4 | Total quality point rating |
| TUE-HR | i1 | Lecture hour Tue 1 is 0800 |
| TUE-LOC | c5 | Location for Tue class 2 bldg 3 room |
| WED-HR | i1 | Lecture hour Wed 1 is 0800 |
| WED-LOC | c5 | Location for Wed class 2 bldg 3 room |
| YEAR | i2 | Year of previous degree |
| YEARGROUP | c2 | Military year group |
| YR-QTR | c3 | Year Qtr of future course rqmt YYQ |

# BIBLIOGRAPHY


1. Date, C.J., An Introduction to Database Systems, Second Edition, Addison-Wesley Publishing Company, 1977

2. Kahn, Beverly K., "A Method for Describing Information Required by the Database Design Process", 1976 SIGMOD, Association for Computing Machinery

3. Martin, James, Computer Data-Base Organization, Prentice-Hall, Inc., 1975

4. Weiderhold, Gio, Database Design, McGraw-Hill Book Company, 1977

5. Kroenke, David, Database Processing, Science Research Associates, Inc., 1977

6. Allman, Eric and others, INGRES Reference Manual, Version 6.0, University of California, Berkeley, 1977

7. Codd, E. F., "A Data Base Sublanguage Founded on the Relational Calculus", p. 35-65, 1971 SIGFIDET Workshop November 11-12, 1971, Association of Computing Machinery, 1971

8. Gillette, Billy E., Introduction to Operations Research, McGraw-Hill, Inc., 1976

INITIAL DISTRIBUTION LIST

| | | |
|---|---|---|
| 1. | Defense Documentation Center<br>Cameron Station<br>Alexandria, Virginia 22314 | 2 |
| 2. | Department Chairman, Code 52<br>Department of Computer Science<br>Naval Postgraduate School<br>Monterey, California 93940 | 2 |
| 3. | Library, Code 0142<br>Naval Postgraduate School<br>Monterey, California 93940 | 2 |
| 4. | Professor Norman F. Scheidewind, Code 52Ss<br>(Thesis Advisor)<br>Department of Computer Science<br>Naval Postgraduate School<br>Monterey, California 93940 | 1 |
| 5. | LCDR S. T. Holl, Code 52H1 (Thesis Advisor)<br>Department of Computer Science<br>Naval Postgraduate School<br>Monterey, California 93940 | 1 |
| 6. | LT COL J. A. Chapman<br>1142 Spruance<br>Monterey, California 93940 | 1 |