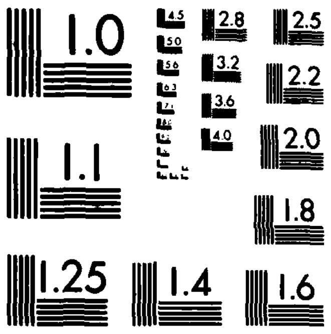
AD-A127 496 INTERACTIVE AUTOMATED SYSTEM FOR NORMALIZATION OF RELATIONS(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING C T TRAVIS 18 MAR 83 1/3

UNCLASSIFIED AFIT/GCS/EE/83M-4 F/G 5/2 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3

INTERACTIVE AUTOMATED SYSTEM

for NORMALIZATION of RELATIONS

THESIS

AFIT/GCS/EE/83M-4 Charles T. Travis
Capt USAF

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY (ATC)

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

83 04 28 113

AFIT/GCS/EE/83M-4

INTERACTIVE AUTOMATED SYSTEM for NORMALIZATION of RELATIONS

THESIS

AFIT/GCS/EE/83M-4 Charles T. Travis
Capt USAF

AFIT/GCS/EE/83M-4

INTERACTIVE AUTOMATED SYSTEM for NORMALIZATION of RELATIONS

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University (ATC)
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

Charles T. Travis
Capt USAF

Graduate Information Systems

18 March 1983

PREFACE

This work presents a user friendly system to define functional dependencies of relations in a relational database and the design and psuedo language coding of the normalization technique to reduce unnormalized relations to Third Normal Form. I feel that with the progress made on this project, the remainder of the implementation can be accomplished either as another thesis effort or as a special study project.

I would like to express my deepest appreciation to Dr. Thomas C. Hartrum, who as my thesis advisor gave me guidance and encouragement. Thanks is also extended to Dr. Henry Potoczny and Major Charles Lillie, who as my thesis readers provided constructive comments to improve the content and clarity of this thesis.

Finally, I wish to thank my wife Sheila and my children Joshua, Dawn, and Reese. They endured endless hours of complaining and separation; however, their patience, understanding, and selfless cooperation enabled me to to complete this graduate program.

CONTENTS

LIST of FIGURES

ABSTRACT

An interactive Automated System for Normalizing Relations was designed and partially implemented with the goal of interacting it with a Relational Database Management System. In addition, this system was to serve as a pedagogical tool for teaching the benefits of normalization for relational database management.

Toward these goals, an extensive literature search and analysis of the six normal forms and other pertinent areas of relation normalization was required in order to identify current issues and areas of research. A main concern was overcome by attempting to locate an algorithm to normalize relations. Most authors present a cursory guide to normalization if any at all, but Jeffrey Ullman presents the concept of "minimal set." If a minimal set is deduced from an unnormalized relation the resultant relations that are formed are in Third Normal Form.

Research was also accomplished in the area of "user friendly" interactive methods. This was needed because the requirement existed for this system to query the user for functional dependencies in order that relations could later be normalized.

INTERACTIVE AUTOMATED SYSTEM for NORMALIZATION of RELATIONS

I. INTRODUCTION

BACKGROUND

Data base technology has been described as "one of the most rapidly growing areas of computer and information science" (Ref 1:63). As a field, it is still relatively young; manufacturers did not begin to offer data base management products untill well into the 1960's. A data base management system (DBMS) can be thought of as a system comprised of a collection of data and a set of application programs which are designed to manipulate the data. An important concept of a database is that the data must be stored in the computer on direct-access devices (such as disks) in order for the computer's central processing unit to be able to utilize the data's cross-references within a reasonable amount of time. In the mid 1970's a different type of data base appeared on the scene. Its name, relational data base, implied that certain data could be viewed as a relation to other data. These relations are described in a two dimensional table consisting of horizontal rows and vertical columns.

In the Digital Engineering Laboratory (DEL) at the Air Force Institute of Technology (AFIT), the need and desire existed for a relational DBMS system, to be used as a pedagogical tool for instructing students in the design, manipulation, and use of database systems. In 1979, 2LT Mark Roth designed and partially implemented a system (Ref 5) which has served as a starting point for the continued development of the AFIT Relational Database System. In addition to the work completed by Roth, 2LT Linda M. Rodgers (Ref 6) continued the implementation of the system in 1982. An additional desire from the DEL was the design and implementation of a stand-alone system to query the Database Administrator (DBA) for functional dependencies of existing unnormalized relations and the subsequent normalization of said relations. This thesis effort is the result of that desire.

Relations in a relational database can be depicted as a two dimensional table consisting of horizontal rows and vertical columns. For example, the relation FIRST, Figure 1, is depicted by a table with 12 rows (tuples) and 5 columns (attributes). This thesis will use the terms "row" and "tuple" interchangably, as well as the terms "column" and "attribute". Each tuple refers to a separate entity, while each column has values that were obtained from the domain of each attribute. In the case of "SNUM", it's domain would contain "S1", "S2", "S3", and "S4". The domain of the attribute "CITY" would contain "LONDON" and "PARIS". So, each attribute has a distinct domain or value set from which its values are drawn.

The relation FIRST has some inherent problems within its structure. The pre[illegible]minant problem is redundancy. The same values for STATUS and CITY exist for all like values of SNUM. For instance, with a SNUM of S1, STATUS is always "20" and CITY is always "LONDON". This redundancy of data is expensive in terms of the additional storage required to hold repetitous data and in terms of the number of updates required if a supplier might move from one city to another. Regarding updates, the relation FIRST would have to be searched to find every tuple with the particular supplier number in question and then the CITY value would have to changed to reflect the new location. This would be costly, as well as cumbersome.

SNUM	STATUS	CITY	PNUM	QTY
S1	20	London	P1	300
S1	20	London	P2	200
S1	20	London	P3	400
S1	20	London	P4	200
S1	20	London	P5	100
S1	20	London	P6	100
S2	10	Paris	P1	300
S2	10	Paris	P2	400
S3	10	Paris	P2	200
S4	20	London	P2	200
S4	20	London	P4	300
S4	20	London	P5	400

FIRST

Figure 1. Sample of Relation FIRST

At this point the term functional dependency (FD) becomes significant. Functional dependency is described as follows: " given a relation R, attribute Y of R is functionally dependent on attribute X of R, if and only if, each X-value in R has associated with it precisely one Y-value in relation R (at any one time)" (Ref 2:240). Functional dependencies might be thought of as a special type of integrity constraint. This means each value of the attribute X in relation R will functionally determine one and only one value for attribute Y in relation R.

With the definition of FD in mind, the main thrust of this thesis comes to mind, that is, normal forms and the normalization of relations. There have been described six normal forms. They are First (1NF), Second (2NF), Third (3NF), Boyce-Codd (BCNF), Forth (4NF), and Fifth (5NF) normal form. The first three forms are stepping stones to BCNF and beyond. The "normalization process" reduces relations to the successive six normal forms, if the relation is not already optimal. Using the predefined FDs, this normalization process produces a collection of new relations that are equivalent to the original relation, but from a data base point of view, more desireable. They are more desireable due to the elimination of redundancy and the creation of compact and meaningful relations.

STATEMENT OF PROBLEM

The purpose of this thesis is to solve two problems. First, a system is to be designed and implemented to query the DBA for functional dependencies of relations that exist in 1NF. This segment of the thesis effort will be a "user friendly" interactive system which will communicate with the database administrator by providing prompts, guidance, accurate display of results, and the ability to negate previous requests.

Secondly, a system is desired to normalize relations to Third Normal Form (3NF). When both goals are combined, the final system will provide the DBA a method to interactively communicate to direct the subsequent normalization of unnormalized relations.

SCOPE

The scope of this thesis is to design and implement an interactive system to normalize relations. The design phase will utilize current Top Down Structured Programming (TDSP) techniques. The main effort will be on the design and implementation of an algorithm to normalize relations that were defined by the Data Definition Language (DDL) (found in Appendix 2) of the Roth database system (Ref 5). In addition, a "user friendly" interface will be designed and implemented to allow the user exclusive control in defining the functional dependencies of the relations to be normalized.

APPROACH

The first step consisted of an extensive literature search to examine the research already completed in the normalization area. A multitude of data base experts have written on the normalization concept. The literature search was then directed specifically to the normalization algorithm area to determine if any research had been accomplished and recorded. Very little work has been recorded in this area. Both Hubbard (Ref 3) and Date (Ref 2) have outlines for algorithms, but do not provide any detailed information towards an algorithm.

Continued search revealed that Ullman (Ref 5:178) proposed the concept of a "minimal set" and then went on to prove that a minimal set of functional dependencies was in 3NF. With this algorithm in mind, the required modules were designed using TDSP methods and then they were compiled and validated, first as stand alone modules and then as an integrated portion of the existing Roth Database System.

SEQUENCE OF PRESENTATION

The remainder of this thesis is divided into seven chapters. Chapter II is the Background chapter. The system configuration is presented, the six normal forms are discussed in detail, and some definitions are examined. Chapter III discusses the System Requirements. Chapter IV is the Top Level Design, while Chapter V is the Functional Dependency Module. Chapter VI is the analysis and design of the normalization technique, the Normalize Module and Chapter VII is the Conclusions and Recommendations chapter.

II. BACKGROUND

This chapter will discuss the system configuration, the Roth System background, and the six normal forms that will be used in this thesis.

SYSTEM CONFIGURATION

This section describes the system configuration associated with the implementation of this thesis. The system configuration section consists of two areas, the machine selection and the operating system.

MACHINE

One of the goals of this thesis is to implement an automated normalization technique on the AFIT Digital Engineering Lab's (DEL) LSI-11s. The LSI-11s are manufactured by the Digital Engineering Corporation as a microcomputer with minicomputer capabilities.

There are five LSI-11s located in the DEL. All software developed from this thesis effort will be executable on each of these machines. Machine configuration and software portability were two of the main reasons for selecting the LSI-11s to accomplish this work.

OPERATING SYSTEM

In order to provide software compatable with previous DEL lab thesis efforts, namely the Roth database system (Ref 5) and the Fonden database system (Ref 6), the decision was made to continue with the University of

California, San Diego (UCSD) version of PASCAL as both the operating system and the implementation language.

PASCAL is a good choice for the implementation language for a number of reasons. First, PASCAL can be structured for easier reading and as a continuation of the top down design effort. Secondly, PASCAL can handle large programs by allowing program segmentation, separately compiled procedures, and virtual memory using segment swapping. Thirdly, PASCAL is used in a multitude of microcomputers, so software systems can be portable by just moving the code to a new computer that has the capability.

ROTH SYSTEM BACKGROUND

The Roth Database System was designed to run on a stand-alone minicomputer under the control of UCSD Pascal Operating System, Version II.0. The present Roth system is being run on the DEL's LSI-11s with a CRT as the CONSOLE device. This section will discuss how this thesis effort interacts with the Roth System.

In addition to the Executive module, hereafter called exec, the Roth System has presently four main logical modules which are in different stages of completion. They are the SETUP module, the DDL Processor module, the DML Processor module, and the SHUTDOWN module. The exec operates in one of two modes: normal mode for user control, and special mode under control of the database administrator (DBA). This thesis effort will only be concerned with the special mode, because normalization of relations modifies the previously defined data definition language, and should only be allowed at the DBA level.

The exec module allows entry to the special mode after the DBA inputs a unique identification password as part of the system logon. Special mode allows the DBA access to the entire database for the purpose of defining, modifying, or deleting domains and relations; for full control of the DDL; as well as the database initialization. The SETUP module, in addition to maintaining the LOGON procedures, also has the responsibility for reading domain and relation definitions from disk and storing them in memory.

The DDL Processor module aids the DBA by creating in memory additional domain and relation definitions as they are input by the DBM. Once defined, these definitions reside in memory, along with any definitions read in by the SETUP module, until the DBA terminates the session, which subsequently causes the SHUTDOWN module to write all the definitions to a disk file on a diskette. These definitions will always be read from disk upon logon to the system, and they will be always written back to disk upon quitting the session.

The DML Processor module controls the execution of all other commands. It's four modules, ATTACH, EDIT, RETRIEVE, and INVENTORY, maintain the responsibility of accessing, modifying, querying, and viewing relations.

ATTACH makes the connection between the user and any desired relations that he/she is authorized to access. EDIT allows the user to modify relations by renaming the relation; inserting, deleting, and modifying tuples within the relation; or changing the relation's password(s). RETRIEVE handles the relational queries by storing, retrieving, and executing the command file where the queries are stored. Also, RETRIEVE processes user's requests to display the contents of the relations. INVENTORY displays to the user a list of domain and relation definitions that currently reside in memory. These definitions could have been originally read from disk, or defined during the current session, or both.

The Roth System also uses the concept of a COMMON area where constant and variable names are defined, constants are assigned values, types are declared, and common procedures are defined. Each system procedure has access to this COMMON area if necessary, so required values or common procedures may be used at will.

This thesis is concerned with actions executable at the DBM level, i.e. normalization, so there will be no further reference to the "normal" mode of the exec or to the options available to "normal" users. The high level data flow of the Roth System modules pertinent to this thesis is depicted in Figure 7. The corresponding structure charts are found in Figure 8. The data flow shows that the Roth Database (DB) program and a SETUP.DATA are required to activate this system. The DB program contains the necessary references to access the compiled code that is required to make this system execute properly. The SETUP.DATA file contains domain and relation definitions that have been defined previously. Once the DB program has been executed and SETUP.DATA has been read into memory, the DBM is ready to accomplish the required actions for the current session.

Within the special mode, the DBA has four upper level options available for use. They are DEFINE, INVENTORY, INITIALIZE,AND QUIT. DEFINE allows the defining of domains and/or relations. INVENTORY displays a list of domain and relation definitions which presently exists in memory. INITIALIZE destroys all existing domain and relation definitions. This is the preparatory command to the DEFINE command if the DBA desires to construct a completely new set of domain and relation definitions. QUIT calls the SHUTDOWN function to write the domain and relation definitions to the output file SETUP.DATA.

SIX NORMAL FORMS

This section will present a background and definition of the terms used in this thesis, as they relate to relational data bases. In addition, six of the various normal forms accorded to relational data bases will be examined. The six normal forms to be covered are First (1NF), Second (2NF), Third (3NF), Boyce/Codd (BCNF), Forth

(4NF), and Fifth (5NF). There have been numerous normal forms suggested, but the previously mentioned ones are the most well accepted. Codd was instrumental in developing 1NF, 2NF, and 3NF. Later, Boyce and Codd set stronger guidance on 3NF and this was to become Boyce/Codd normal form (BCNF). Fagin later developed both 4NF and 5NF (Ref. 2:238-9).

RELATIONAL DATABASE CONCEPTS

A relation consists of an unordered set of entries. Each entry is a complete and meaningfull collection of related information about the objects around which the relation was composed. For instance, if the relation contains information that pertains to suppliers for a certain job, then each entry (tuple) would correspond to a specific supplier. There would be an entry for each supplier for the job. The tuples are composed of fields called attributes. There would be an attribute for each of the following: supplier name, supplier number, city, part number, and quantity supplied.

KEYS

An important concept to relational data bases is the notion of keys. Keys are the foundation to understanding and using relational data bases. It is frequently found within a given relation there is a "collection" of one or more attributes with values that can uniquely identify each tuple within the relation. This is called the primary key.

In the supplier example, a supplier number would uniquely identify tuples (suppliers) within the relation. The supplier number attribute is said to be the primary key for this relation. Not all primary keys are single valued. In this case it was, but numerous relations will have a combination of two or more attributes as primary keys. By default, this primary key is also what is called a "candidate" key, that is, a key that identifies each tuple, but it may or may not be the primary key.

Some relations might have more than one attribute combination possessing the unique identity property, thus, more than one candidate key. The previously mentioned supplier relation is a possibility, with each supplier having a unique supplier number and a unique supplier name. At this point, one of the candidate keys must be arbitrarily chosen as primary key for the relation, with the other candidate key being an alternate key.

Up to this point, primary keys have been noted to identify tuples. This is true, but the tuples actually are entities that exist in the real world. Because of this, there exists a rule that no component of a primary key value may be null. This is because each entity must have an identifier. So, a null identifier, which corresponds to nothing, is not allowed.

A second rule states, when an tuple of a relation references a tuple of a second relation, that tuple has to exist in the second relation. Otherwise there would be a reference to a nonexistent tuple.

Two important properties of candidate keys are as follows:

* Each attribute of a given relation is functionally dependent on each candidate key of the relation in question.
* The attributes of a candidate key is a maximal funcally independent set, i.e., every proper subset of the attributes within the primary key is functionally independent of every other proper subset of the attributes within the primary key, and no other attribute from the relation can be added without destroying this functional independence.

Any candidate key of the relation can be designated as the primary key of the relation. This was accomplished on the supplier data base when a primary key was selected from the two available candidate keys. The importance of designating a primary key of a relation is to assure that there are values placed in the primary key's attributes for each entry in the relation. The remaining attributes can be empty at the time an entry is created and can be filled in later.

FIRST NORMAL FORM

An attribute may consist of a single discrete value, or a set of values. In the case of single direlation
in question.

* The attributes of a candidate key is a maximal funcally independent set, i.e., every proper subset of the attributes within the primary key is functionally independent of every other proper subset of the attributes within the primary key, and no other attribute from the relation can be added without destroying this functional independence.

Any candidate key of the relation can be designated as the primary key of the relation. This was accomplished on the supplier data base when a primary key was selected

from the two available candidate keys. The importance of designating a primary key of a relation is to assure that there are values placed in the primary key's attributes for attribute of a relation in 1NF can be a relation itself. A table composed of two or more entries or two or more attributes is referred to as a relation as defined in Chapter 1.

To convert an unnormalized relation to 1NF, each attribute that consists of multiple values would be examined to see which of the two following cases apply. In the case of a single entry relation, each of its sub-attributes would be added to the original relation, renaming attributes as necessary, to avoid similiar or confusing names. Or, if an attribute contains multiple entries, then this attribute entry should be established as its own relation in the methods described earlier.

SECOND NORMAL FORM

A relation is in 2NF if and only if the following conditions hold:

1. The relation is in 1NF.

2. Every attribute not a member of a candidate key is fully functionally dependent on each candidate key of the relation.

An example of a relation being in 1NF, but not in 2NF is the relation FIRST presented in Chapter 1, Figure 1. As stated before, FIRST has problems of redundancy and insertion/deletion conflicts. To eliminate some of the problems, FIRST can be normalized into 2NF as depicted in

Figure 2.

The solution to the problems of FIRST is to form two new relations, SECOND and SP, and ignore FIRST from any further consideration other than for historical purposes.

SNUM	STATUS	CITY	PNUM	QTY
S1	20	London	P1	300
S1	20	London	P2	200
S1	20	London	P3	400
S1	20	London	P4	200
S1	20	London	P5	100
S1	20	London	P6	100
S2	10	Paris	P1	300
S2	10	Paris	P2	400
S3	10	Paris	P2	200
S4	20	London	P2	200
S4	20	London	P4	300
S4	20	London	P5	400

FIRST

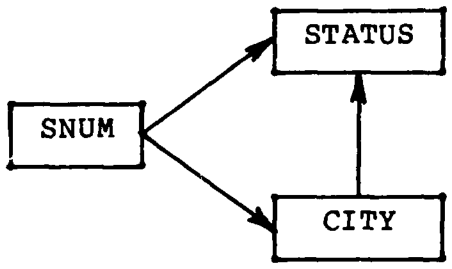
Figure 1. Sample of Relation FIRST (Ref 2:244)

By comparing the relation FIRST against SECOND and SP, it can be easily seen that the final effort was to eliminate the non-full functional dependencies, which was the answer to the previous problems. Upon closer examination, one might determine that problems in 1NF relations, but not in 2NF relations, exist due to the mixing of two types of information (Ref. 1:246). In this case, city and status information was combined with quantity information.

The question surfaces of how were SECOND and SP derived from the relation FIRST? This example worked out to

SNUM	STATUS	CITY
S1	20	London
S2	10	Paris
S3	10	Paris
S4	20	London
S5	30	Athens

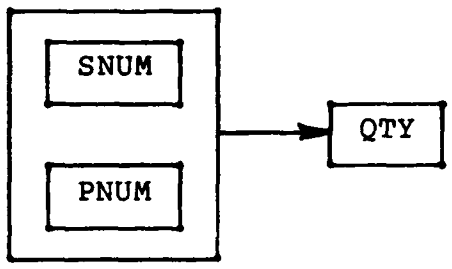
SECOND



FDs of SECOND

SNUM	PNUM	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	300

SP



FDs of SP

Figure 2. Relations and FDs of SECOND and SP (Ref 2:245)

be quite simple by projecting out sets of attributes in accordance with the functional dependencies which were stated earlier. The first step was to pick a pimary key for the relation FIRST. This could be done arbitrarily, but since the supplier number by itself could tell the status and city of the tuple, while the combination of both the supplier number and part number were required to determine the quantity of a part, the supplier number was selected as the "primary key". Thus SECOND evolved as a relation with the functional dependencies of SNUM ---> STATUS, SNUM ---> CITY, and CITY ---> STATUS. In the meanwhile, a relation called SP was formed with its key as SNUM and PNUM, and the functional dependency of (SNUM, PNUM) ---> QTY.

Another question comes to mind. What if the original relation structure is needed later? It can be recovered by taking a natural join of the relations SECOND and SP, thus giving back the relation FIRST. This is called a lossless decomposition, which means any information that could be obtained from the original relation can also be obtained from the two new relati' ıs.

THIRD NORMAL FORM

The relations SECOND and SP have eliminated the problems of redundancy, but there still exists a problem with the mutual independence of SECOND's non-key attributes. In other words, there are non-key attributes dependent on the key thru another attribute. This is called transitive dependency. Third Normal Form relations do not contain this transitive dependency, hence this redundancy of dependencies is desireably absent.

For example, it can be seen that two dependencies exist in the relation SECOND, "SNUM -- -> CITY" and "CITY - - - > STATUS". By observation, it can also be seen that SNUM ---> STATUS, thru CITY. This is an implied, or transitive dependency. This transitive dependency is forbidden if a relation is to be in 3NF. In fact, in order for a relation to be in 3NF, it has to be in 2NF and contain no transitive dependencies between the attributes. This transitivity leads to difficulties in updating, inserting, and deleting, so it should be eliminated.

In order to eliminate the transitive dependence of STATUS on SNUM, the relation SECOND has to be projected out into two new relations, i.e., SC(SNUM,CITY) and CS(CITY,STATUS). Figure 3 shows the relations SC and CS, with the appropiate functional dependency of each. At this point, the relations SC and CS are in 3NF, while SECOND was not.

What has been shown is that a relation in 2NF, but not 3NF, can be put into 3NF by projecting out an equivalent set of relations. This process is reversible by performing a "join" between the two relations on the attribute, in this case CITY, that is common to both of the new relations.

For the data base administrator, 3NF is an aid to precise thinking. A data base in 3NF can grow and evolve naturally. The updating rules are straight forward, where tuples (records) can be added or deleted without the problems that occur in relations with non-3NF tuples. The two new relations previously formed are operatonally better than SECOND because now there can be city and status information for cities where there is no supplier. Third normal form structuring gives a simple view of data to the programmers and users, and makes them less likely to perform invalid operations.

SNUM	CITY
S1	London
S2	PARIS
S3	Paris
S4	London
S5	Athens

SC

$$\begin{tikzcd} \boxed{\text{SNUM}} \arrow[r] & \boxed{\text{CITY}} \end{tikzcd}$$

FDs of SC

CITY	STATUS
Athens	30
London	20
Paris	10

CS

$$\begin{tikzcd} \boxed{\text{CITY}} \arrow[r] & \boxed{\text{STATUS}} \end{tikzcd}$$

FDs of CS

Figure 3. Relations and FDs of SC and CS (Ref 2:248)

BOYCE/CODD NORMAL FORM

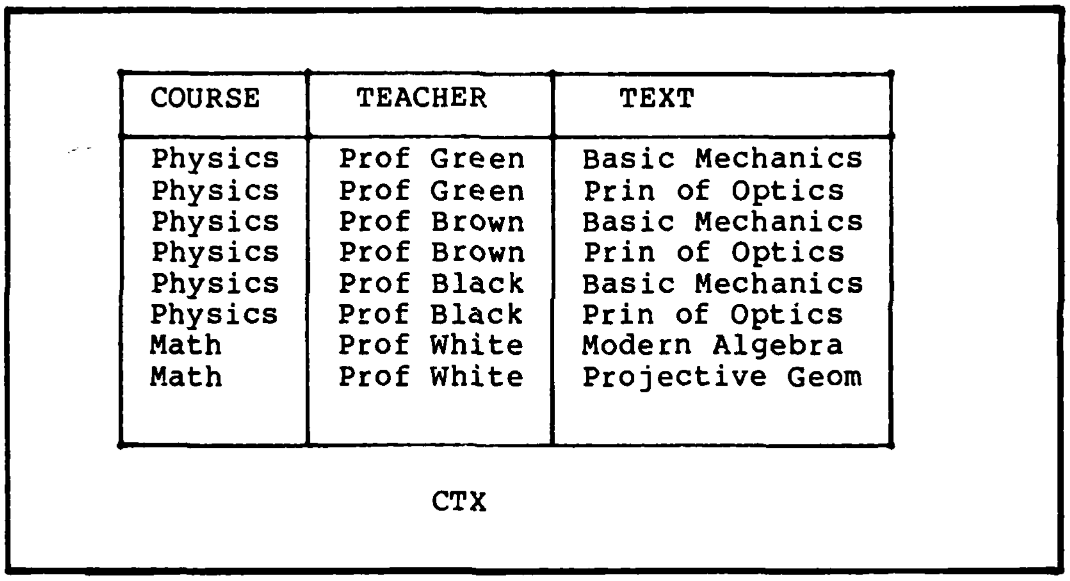
When Codd originally developed the concept of 3NF, he did not take into account the fact that a relation might have more than one candidate key. As earlier stated, a candidate key is a key made up of a combination of attributes. Each candidate key possesses unique identity properties that functionally determine all the attributes within the relation. The 3NF definition was replaced by a stronger, but conceptually simpler, definition that was due to the efforts of Boyce and Codd. It was to be known as Boyce/Codd normal form (BCNF).

BCNF does not make any reference to 1NF, 2NF or transitive dependencies. But, BCNF does define the term "determinant", as a set of attribute(s) on which some other attribute(s) is/are fully functionally dependent. This brings the discussion to the definition of BCNF. A relation is in BCNF if and only if every determinant is a candidate key. That is to say, every determinant fully functionally determines all the attributes within the relation. If not, the relation should be projected out into two or more relations that fullfill this requirement.

Ullman (Ref. 4:189) has done some additional reseach in the normalization concepts. One of his conclusions is that any relation in BCNF is also in 3NF. He also implies that BCNF may cause the loss of some previously defined dependencies.

FORTH NORMAL FORM

If there exists a relation CTX that contains information concerning courses, teachers, and texts used by the teachers in the particular courses, it is easy to see the problem of redundancy reoccurring. This is shown in Figure 4. Since one of the goals of normalization is to eliminate, as much as possible, any redundancy, this subsection will examine methods along this thought.



COURSE	TEACHER	TEXT
Physics	Prof Green	Basic Mechanics
Physics	Prof Green	Prin of Optics
Physics	Prof Brown	Basic Mechanics
Physics	Prof Brown	Prin of Optics
Physics	Prof Black	Basic Mechanics
Physics	Prof Black	Prin of Optics
Math	Prof White	Modern Algebra
Math	Prof White	Projective Geom

CTX

Figure 4. Relation CTX (Ref 2:256)

The relation CTX is in BCNF, since the whole relation is made up of keys and there exist no additional functional determinants. This type of problem exists in BCNF relations because when related data is "grouped" together, the data might not be dependent on each other. In this case, TEACHERS and TEXTS are independent of each other.

Attributes of CTX cannot be projected out to form "new" relations, based on functional dependencies. This is because the relation is a key itself. However, two new relations can be created if a different type of dependency is considered. This type is called multivalued dependency (MVD). A MVD exists when a value of an attribute can determine more than one value for the dependent attribute. For example, a male human could have been married to a number of wives, with each union producing a number of offspring. Each of these would be a MVD. The definition of MVD is as follows:

> Given a relation R with attributes A, B, and C, the multivalued dependence
>
> R.A --> --> R.B
>
> holds in R if and only if the set of B-values matching a given (A-value,C-value) pair in R depends only on the A-value and is independent of the C-value. As usual, A, B, and C may be composite (Ref. 2: 258).

This definition will only hold in relations that have more than three attributes.

A functional dependency (FD) is a special case of a MVD. In fact, a FD is a MVD in which the set of values consists of a single value.

The definition for 4NF (Ref. 2 : 259) states:

> A relation R is in 4NF if and only if, whenever there exists an MVD in R, say A -->B, then all attributes of R are also functionally dependent on A.

If the relation CTX is re-examined, it is obviously not in 4NF. This is because when COURSE --> --> TEACHER, there is not an FD between COURSE and TEXT. In order to place CTX in 4NF, it is necessary to project out attributes according to the stated MVDs. This produces two relations, CT and CX, as shown in Figure 5. These structures are more desirable than BCNF, because of the elimination of the redundancy due to improper structure. Fagin says " any relation can be nonloss-decomposed into an equivalent collection of 4NF relations" (Ref. 2 : 259). What is not stressed is that it might be undesirable to decompose a relation into 4NF. This was also true with BCNF too.

COURSE	TEACHER
Physics	Prof Green
Physics	Prof Brown
Physics	Prof Black
Math	Prof White

CT

COURSE	TEXT
Physics	Basic Mechanics
Physics	Prin of Optics
Math	Modern Algebra
Math	Projective Geom

CX

Figure 5. Relations CT and CX (Ref 2: 257)

FIFTH NORMAL FORM

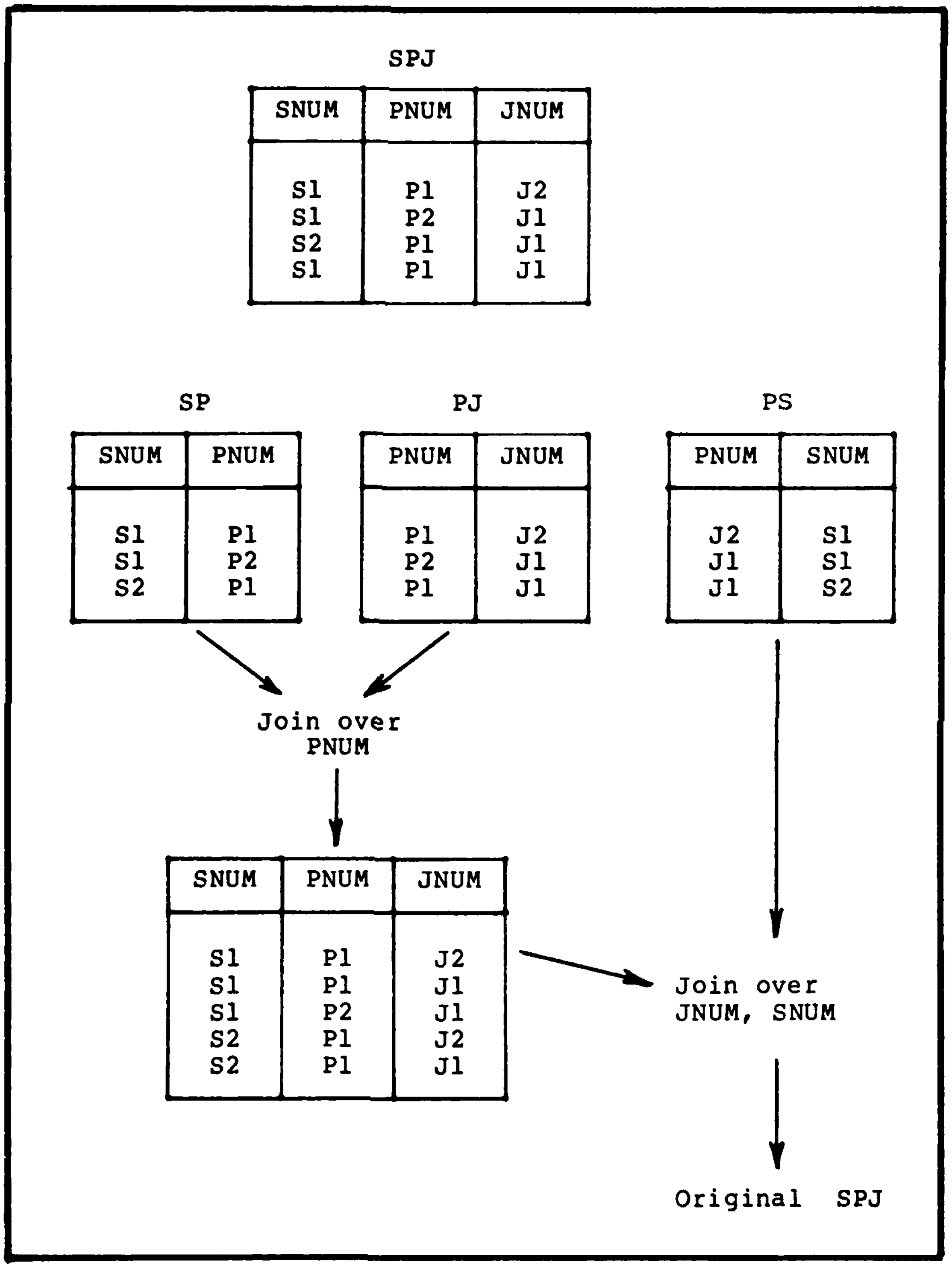
Further research into normalization unexpectedly discovered that there exist relations that can not be nonloss-decomposed into just two relations, but can be satisfactorily nonloss-decomposed into three or more relations (Ref 2 :260). This is a very special type of relation that contains at least three attributes, where all are keys and there exists no non-trivial FDs or MVDs. This is shown in Figure 6, where the relation SPJ cannot be projected non-losslessly into two new relations, but it can form three new relations, SP, PJ, and JS, without losing its integrity.

Figure 6 also shows that by joining any two of the sub-relations over their similiar attribute, an intermediate relation (with three attributes) is formed. This intermediate relation then can be joined with the third sub-relation over the attributes of the third relation to produce the original relation in its entirity. This shows that it has been a nonloss-decomposition.

SUMMARY

Normalization is a powerful tool for the relational data base manager. The main benefit is the elimination of redundancy within the relation. This will reduce both the space requirement to store relations, as well as aid in the insertion/deletion problems. But normalization has its limits. Of the six normal forms described in this chapter, only the first three (1NF, 2NF, 3NF) are 100% beneficial.

The others (BCNF, 4NF, 5NF) can produce relations which have lost some of their original dependency meanings, so are of less benefit.



SPJ

SNUM	PNUM	JNUM
S1	P1	J2
S1	P2	J1
S2	P1	J1
S1	P1	J1

SP

SNUM	PNUM
S1	P1
S1	P2
S2	P1

PJ

PNUM	JNUM
P1	J2
P2	J1
P1	J1

PS

PNUM	SNUM
J2	S1
J1	S1
J1	S2

SNUM	PNUM	JNUM
S1	P1	J2
S1	P1	J1
S1	P2	J1
S2	P1	J2
S2	P1	J1

Figure 6. Join of three projections, back to the original. (Ref 2:260)

III. SYSTEM REQUIREMENTS

This chapter discusses two main areas. They are presented in the following order: user requirements and assumptions for this thesis.

USER REQUIREMENTS

Since the main objective of this thesis is a product that will aid the DBA in his/her day to day job, the following requirements are hereby established:

1. Develop a user friendly system.
2. The system must be interactive with the user.
3. Will be an automated method to normalize relations.
4. Will complement the existing Roth Database System.
5. The thesis effort will be a stand-alone system which will access the data file output by the Roth Database System.
6. The output file of this thesis effort will be compatible with the Roth Database System.
7. Normalize relations to at least 3NF.

Each of these requirements are necessary for a completely viable system.

ASSUMPTIONS

The following assumptions are in effort for this thesis effort:

1. Each relation is treated as an individual.
2. Functional dependencies will only exist within a relation, not between relations.

3. The following will be required to normalize a relation:

    a. Name of the relation.

    b. Attributes of the relation.

    c. Functional dependencies within the relation.

4. Relations required to be created during the normalization process will be created in accordance with Appendix A, section 3.4.A, of the Rodger's followup to the Roth Database System (Ref 5).

These assumptions are required for an orderly appraisal of the effort and restrictions required for this thesis.

IV. TOP LEVEL DESIGN

INTRODUCTION

This chapter will outline the top level design for this thesis effort. Since this thesis will complement the Roth Data Base System (Ref. 5), the integration with the Roth System will also be examined in this chapter. In addition, the top level data flow graphs and structure charts of the Roth System and of the necessary modifications to this system will be discussed.

The objective of this thesis is the design of a user friendly, interactive system that queries the user for functional dependencies of an existing relation and then normalizes the the relation. This thesis was accomplished in two steps. First, the user friendly functional dependency section was designed, coded, and implemented. Then the normalization section was designed, coded, and implemented.

Each step outlined above was treated as a separate segment because, while they worked together in achieving the end product, they performed totally independent tasks. The important idea to remember is that while the two segments are somewhat independent of each other, the normalization segment does require the functional dependency segment to have created functional dependencies before the relations can be normalized.

SYSTEMS INTERACTION

Based on an understanding of the Roth System as described in Chapter II, the interaction of this thesis effort with the Roth System will be examined. This thesis effort is intended to execute as a stand-alone system. It will need access to the data file SETUP.DATA in order to perform normalization on the needed relations. The Roth System itself will not be required, but some of its modules have been extracted to insure consistant input and output capabilities.

The Roth System modules that can be used (but need to be modified) include the SETUP module, the QUIT module, the DEFINE module, and the COMMON unit. The required changes are in the following paragraphs.

The present Roth Database System COMMON module defines each relation to be a record with the following: a name, a pointer to the next relation in the list, a pointer to a separate list of attributes, a pointer to a separate list of key attributes, a security record, a tuple count, a filer name, and some special purpose boolean switches.

In order to determine if a specific relation needs to be normalized or not, a boolean switch must be added to each relation record structure. Then the proposed system would only have to traverse the linked list of relations and check each record's NORMALIZE switch to determine if the relation has already been normalized. If not, the necessary procedures will be called to normalize the relation. If by checking the switch, it was determined the relation did not need normalization, then the traversal of the linked list would resume until the next unnormalized relation is located or the end of the list reached.

The Roth System's data file SETUP.DATA will have been stored on a disk file and contains all prevously defined domain definitions and relation definitions. The difference between the Roth SETUP.DATA and the SETUP.DATA created by this thesis effort is the addition of a NORMALIZE switch which has been inserted into each relation's definition in the COMMON unit and subsequently written out to disk.

The current SETUP module first reads the domain definitions from the input file SETUP.DATA and stores them in memory. Then SETUP reads the relation definitions from the input file and stores them in memory. Each set of definitions are stored as a linked list with a head pointer value returned so the system can access the lists. The SETUP module requires modification to read the domain definitions straight from the input file to the output file. This is because the domain definitions are not needed in this thesis effort, but they are required to remain on the SETUP.DATA file for use by the Roth System itself. The SETUP module must also be modified to read in the added boolean NORMALIZE switch in the relation records. This is necessary because this thesis program will need to examine each relation's NORMALIZE switch to determine if the relation is in a normalized form or not.

The module QUIT must be modified in a corresponding manner. That is, since the domain definitions will already be written on the output file by the modified SETUP, the code in QUIT that normally would write them out can be deleted. Since the proposed system requires an interlaced method of reading and writing to and from disk, it was determined that a totally separate output file would be incorporated in this system. This would even allow comparison between the "before" and "after" data files. Obviously, the output file will have a different name than the input file, in order to make them distinguishable to the PASCAL filer system. QUIT module should also be modified so when it is writing out the relation definitions to the output data file, it also writes out the value of the NORMALIZE switch.

As stated before, the DEFINE module allows the defining of domain and relation definitions. This is performed by querying the user for names and other characteristics of the domains and relations after the user has indicated the desire to define either domains or relations. DEFINE must be modified so it will set the added NORMALIZE switch to "off" or "false" to indicate that the relation is unnormalized. The switch will remain in this "false" state until the normalization program is executed and this relation itself is normalized. When normalized, the relation's NORMALIZE switch will be set to "true".

During the design stage, the question arose as to whether the DBA should be allowed to set the NORMALIZE switch for relations that he desired to remain in the unnormalized state. Since the thesis effort involves the implementation of a system to "automatically" normalize relations, the decision was made to not allow the DBA to set the normalize switch. This will keep all relations consistant with each other in that they all will be examined for normalization. The point to be made is that the DBA might input functional dependencies in such a manner that the relation is already in 3NF. But, each relation will be examined to determine if it needs normalization or not.

In summary, the modules that required modification for the Roth System were the COMMON unit, the SETUP module, the QUIT module, and the DEFINE module. The COMMON unit had the addition of the normalize switch in each relation record, as well as the additional record structure to facilitate the functional dependencies as they are defined. The changes to the SETUP, QUIT, and DEFINE modules centered about the reading, writing, and defining of relations and the necessity to handle the normalize switch appropiately.

The Roth System modules which were used in this thesis effort includes the COMMON unit, the SETUP module, and the QUIT module. The COMMON unit is exactly the same as the modified COMMON described in the previous paragraph. The SETUP module must be changed so it will immediately write domain definitions to the output file as they are being read from the input file SETUP.DATA. SETUP will still write the relation definitions to memory as it reads them from disk, including the normalize switch for each relation. The QUIT module must be changed so it will correspond to the SETUP module. It will not have to write the domain definitions out to memory because SETUP would have already accomplished this task, but QUIT will still need to write out to the output file the relation definitions which includes the normalize switch.

HIGH LEVEL DATA FLOW

The high level data flow diagram for this thesis effort is found in Figure 7. The structure chart is found in Figure 8. The data flow diagam shows that the user is queried for the input and output file names. The input file should be SETUP.DATA, but at a minimum it must contain data in the format as it would be found in SETUP.DATA. Because domain definitions are written to the output file as they are read in from the input file, totally separate and distinct input and output files must be used. The system will create the actual output file with a user supplied name. The new output file can later be renamed SETUP.DATA for use by the Modified Roth System .

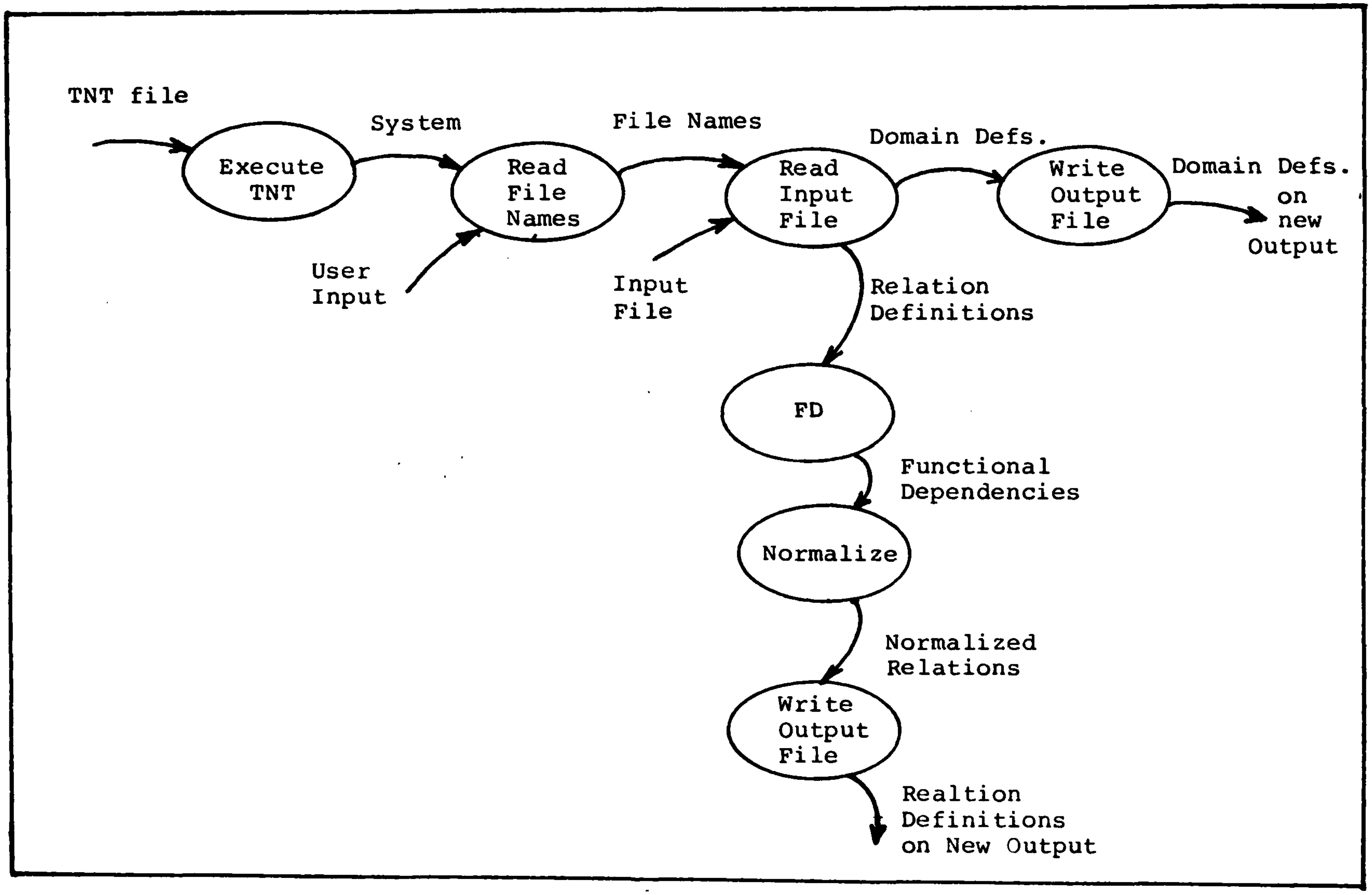


Figure 7. Top Level Data Flow for this Thesis

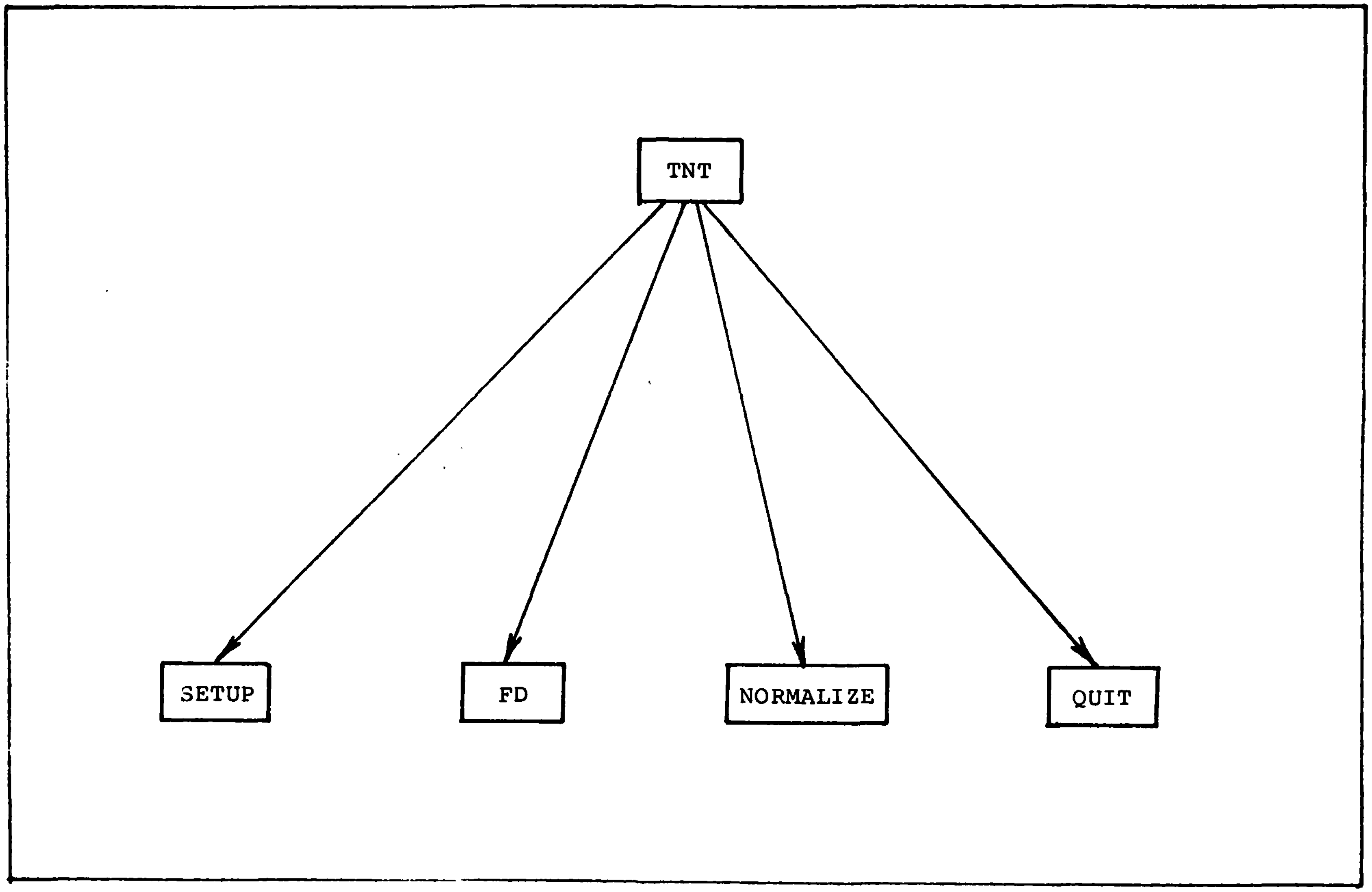


Figure 8. Top Level Structure Chart for this Thesis

Upon accepting the input and output file names from the user, the system should then read the input file (usually SETUP.DATA). The domain definitions will be written to the output file immediately upon being read. The relation definitions will subsequently be written to memory after they are read from the input file.

This thesis effort has been designed around two major concepts. They are describing functional dependencies for unnormalized relations, and then normalizing relations as needed. The two major modules in this thesis effort will be FD and NORMALIZE. The module FD is concerned with determining functional dependencies. Through user friendly interactive queries, FD will write into memory the functional dependencies of each unnormalized relation, as the user supplies them.

Once FD has determined the functional dependencies, the module NORMALIZE will then normalize the relation in question to Third Normal Form (3NF). This means that each subsequent relation generated will be in 3NF and its normalize switch will be set to "true". When breaking each unnormalized relation into new relations, the user will be queried to supply names for the new relations.

When all unnormalized relations have been normalized, the module QUIT will write the relations to the output file. The system will then sign off with an ending message to let the DBA know that the system has completed execution.

ROTH SYSTEMS MODIFICATIONS

In order to generate a new SETUP.DATA compatable with this thesis effort, the Roth System will have to undergo some modifications. The changes are necessary due to the addition of the "normalize" switch in each relation record. Each of the modules that accesses the linked list record structure will require the modification. SETUP and QUIT are required to read and write the relations to and from disk, so changes will be needed in each. DEFINE will also need to be changed to account for this added feature. It will set all newly derived relation's normalize switch to false, so they will be examined for normalization.

In addition, the COMMON module for the modified version of the Roth Database System will require the same changes proposed for the thesis effort. In fact, the same COMMON module can be used for both systems.

CONCLUSIONS

This thesis effort will generate a system that will manipulate unnormalized relations by first determining functional dependencies; and then by reducing the relation in question into new relations that are in Third Normal Form. The first step of this thesis was to modify the Roth System so relations in COMMON would contain a "normalize" switch and the needed extra record structures. The second step of this thesis was to modify other Roth modules so they would read and write to SETUP.DATA the added normalize value. Thirdly, a system was then devised to perform the derivation of the functional dependencies of unnormalized relations. Lastly, modules were designed and implemented to normalize the needed relations.

V. FUNCTIONAL DEPENDENCY (FD) MODULE

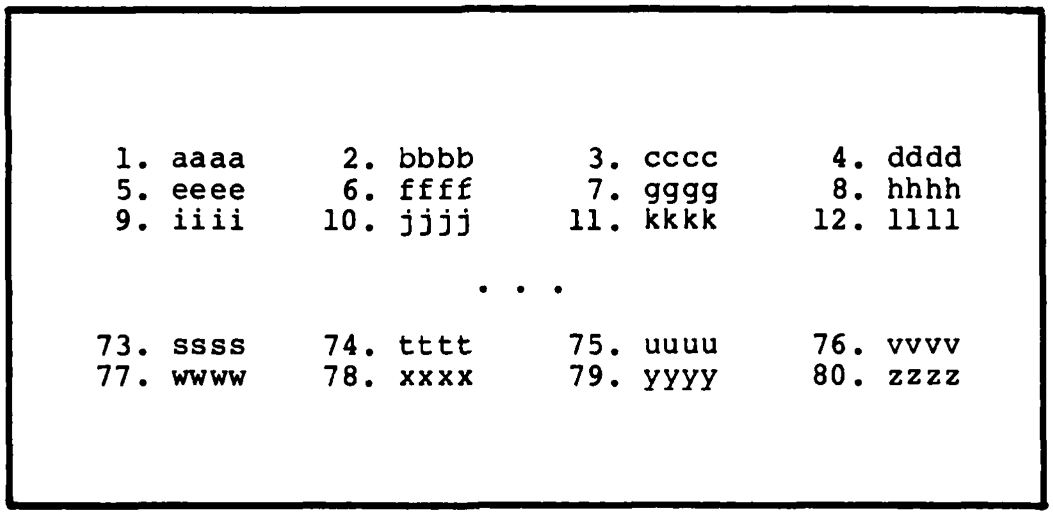
INTRODUCTION

Relational data bases are becoming popular in their use throughout the world of information systems. A common fallacy of relational data bases is the use of unnormalized relations within the data base. This can cause a tremendous volume of duplicated data which can be eliminated by normalization. In order to normalize each relation, the functional dependencies (FDs) of the particular relation have to exist in some form. This chapter will examine the segment of this thesis effort that produces the FDs.

FD MODULE DESIGN

The executive module TNT is designed to traverse a linked list structure of relations in memory searching for unnormalized relations. When such a relation is found, TNT calls the module FD so that functional dependencies can be input by the Database Administrator (DBA). The module FD is passed a value that points to a relation that needs to have its functional dependencies defined.

FD is designed to ask the DBA to input the attributes which comprise the determinant attribute set and the attributes that comprise the dependent attribute set of individual functional dependencies. To aid the DBA in knowing exactly which attributes are in the relation in question, a menu display of the attributes was determined to be useful. Each attibute is displayed with its key number in front of the attribute. The key numbers run sequentially from 1 to N (with N being the number of attributes in the relation). The attributes are numbered in a left to right fashion with attributes 1 to 4 on the first line, attributes 5 to 8 on the second line, and so on until all attributes or 20 lines have been displayed. If there exist more attributes than have been displayed, the DBA signals for a continuation of display by pressing the <RETURN> key. Figure 9 depicts the screen display which will be presented to the DBA so he may select attributes to define functional dependencies.



```
 1. aaaa    2. bbbb     3. cccc     4. dddd
 5. eeee    6. ffff     7. gggg     8. hhhh
 9. iiii   10. jjjj    11. kkkk    12. llll

                  . . .

73. ssss   74. tttt    75. uuuu    76. vvvv
77. wwww   78. xxxx    79. yyyy    80. zzzz
```

Figure 9. Sample Screen Display of Attributes.

Functional dependencies, as implied earlier, consist of two sets of attributes: the determinant set and the dependent set. The determinant set consists of attributes that imply (determine) the dependent set of attributes. FD first asks the DBA to input individually the attributes that comprise the determinant set. Once the DBA has decided on an attribute to input, he is required to type in the key number of the attribute and press the <RETURN> key. This continues until all determinant attributes have been input, then the DBA answers "no" to the query asking if there are more determinant attributes to be input. Then FD asks the DBA to individually input the attributes of the dependent set. This is done exactly as was done with the determinant attributes, i.e., input key number, press <RETURN> key, repeat these two steps as needed, then answer "no" to exit the input cycle.

In order to allow the DBA to check and recheck his work, a series of acceptance queries is presented to the DBA. This series consists of the ability to accept or reject the whole determinant set, accept or reject the whole dependent set, or accept or reject the whole defined functional dependency. A query is presented after each step has been completed. For instance, after all determinant attributes have been input the DBA indicates no further attributes are to be input, then a listing of the attributes will be displayed and the DBA will have the opportunity to accept or reject the whole set of selected attributes. If rejected, the DBA will be required to input the entire set of determinant attributes again (corrected version). If accepted, the DBA is then requested to input the dependent set of attributes. When accomplished, a similar query will allow the DBA to accept or reject the entire dependent set. If the dependent set is rejected, the DBA is then required to input the corrected set. If accepted, the next step is to accept or reject the entire functional dependency. FD displays the entire functional dependency and the follow-on accept/reject query. If the whole functional dependency is rejected, internally the functional dependency is destroyed and the DBA is asked to begin again by first naming the determinant attribute set and then continuing on as before until an acceptable functional dependency has been input. If the functional dependency is accepted, FD has been completely executed and control is passed back to the executive module TNT.

"USER FRIENDLY" DESIGN CONSIDERATIONS

Since the module FD is the main interface with the DBA, it was desired that FD would be as "user friendly" as possible. To this end, the design of FD was accomplished with four main considerations in mind. The considerations contemplated were human factors, software, hardware, and the required applications. But even before these, there are some overall observations that should be reviewed . They are the concepts of design consistency, design standards, and design tradeoffs.

Design consistency is the first cardinal rule of all design activities. Consistency is important because it can reduce requirements for human learning by allowing skills learned in one situation to be transfered to another situation like it. While any new automated system must impose some learning requirements on its users, it should avoid burdening productive learning with nonproductive, unnecessary activity. Inconsistencies in design are caused by differences in designers, as well as from pressure imposed by time constraints. The solution in these cases usually consists of exceptions that the user must learn to handle. People percieve a system as a single entity. To them it should look, act, and feel similar thoughout. Excess learning can hinder their performance and ultimately influence their acceptance of the system.

The module FD is an interactive module that queries the DBA for functional dependencies. FD displays on the screen the name of the relation and a menu (listing) of its attributes, each with an associated key number. FD first asks the DBA to name the determinant attributes, and then FD asks the DBA for the dependent attributes. The DBA is required to input each set of attributes by keying in the associated key number of the first attribute, then the number of the second attribute, and so on until all attributes of the determinant set have been input. Then all attributes of the dependent set will be input in the same fashion. The querying for determinant and dependent attributes have been consistently designed. Each section of the module that requests the naming of attributes do themselves produce queries, accept responses, and display error notices similarly and consistently. Because these two sections provide similar services, consistency and procedural usage are not difficult to substantiate.

Design consistency is achieved by applying design standards. The purpose of design standards is to provide a product that is (1) consistant from both an appearance and a procedural usage standpoint, and (2) visually clear and easily used (Ref 7:32). Their objective is to reduce the subsystem processing errors and increase processing speed by faster initial learning of screen formats and discouraging "extraordinary situations" during daily activities. Some people might say that designer creativity may suffer from the imposition of design standards, but that would seem a small price to pay for an effective design.

As stated before, the module FD is consistant in its outward appearance (to the DBA) and its procedural usage. The sections that process queries, responses, and error notices are also easily used by the DBA. In fact, because of the display of a menu of attributes, the DBA is required to input only the associated number of each desired attribute, not the whole name, which would have allowed spelling errors, confusion, and frustration.

Design tradeoffs are products of: incompatible designer guidelines; time, accuracy, cost, or ease-of-use requirements; and human requirements. Design guidelines often conflict with one another or with machine processing requirements. In such conflicts the designer must weigh alternatives and reach a decision based on accuracy, time, cost, and ease-of-use requirements. This leads to another rule in user friendly systems designs: Human requirements always takes precedence over machine processing requirements. While it might be easier for the designer to develop a system at the expense of the users, this must not be tolerated.

The module FD and its submodules were designed with an emphasis on ease-of-use concepts. For instance, a menu of attributes for the relation in question will be displayed with a key number in front of each attribute. When required to input attributes, the DBA will only have to input the associated key number, not the whole attribute name. The DBA also has the the choice of continuing or stopping the current session each time a relation has had its functional dependencies input and the relation has been subsequently normalized. This allows the DBA the choice to curtail the current session, which causes all the relations (whether normalized or not) to be written to the new output file. At a later session the DBA might re-execute the TNT system using the last output file as the new input file. In the new session the DBA has the same choice of sequentially handling as many unnormalized relations as he wants to or as time permits. Time savings and convenience will aid the DBA in accomplishing his objectives for the session.

Human considerations are the needs and requirements of the user and are oriented toward clarity, meaningfulness, and ease-of-use. Ease-of-use has been a constant goal in the design of this thesis project, as has been discussed in previous sections. Hardware and software considerations reflect the physical constraints of the terminal on which the screen will be used and the characteristics of the controlling program. They provide a framework within which the screen design must occur. The CRT display screens used with the LSI-11s in the DEL lab are limited to an 80 column by 24 line display, so it was felt that when trying to display menus of attribute names that a maximum of four columns with a maximum of 20 lines of attribute names would not degregate the readability of the screen.

Roth originally decided to allow attribute names to be up to 132 characters in length, but it was felt necessary to truncate the attribute names to the first 14 characters in order to allow the display of four columns on the screen. This truncation is only in effect for the display of the attribute names. The names remain unchanged in memory. The module SHOW_ATRIBS displays the names in the manner discussed in the previous paragraph. Later in this chapter, SHOW_ATRIBS will discuss in more detail the screen display of the attribute names.

If the relation has more than 80 attributes, two or more screens of display will be required. After a screen has been displayed, an automatic pause will allow the DBA to examine the list as necessary. When ready, the DBA will signal for a continuation of the listing by pressing the <RETURN> key. Then the next set of attributes will be displayed either until the screen is full or the last attribute has been displayed. Application considerations reflect the objectives of the system for which the screen is being designed. They are the data or information building blocks which make up a screen display (Ref 7:14). The application consideration for FD is the input of accurate functional dependencies that will aid the system to correctly normalize relations.

FD MODULE IMPLEMENTATION

The module FD is depicted by its structure chart found in Figure 10. FD was implemented in a top down structured method. First, the upper level FD module was designed, coded, and implemented. Then each of the lower level modules were individually designed, coded, and implemented. Stubs were used in FD where the lower level modules would eventually be called. As each lower level module was implemented, the previously mentioned stubs were replaced with the modules themselves.
In addition to the FD module, there are four (4) lower level modules that complement this subsystem. The lower level modules are called SEEK_ATR_NAME, SHOW_ATRIBS, DETER_ATRIBS, and DEPEND_ATRIBS.

SEEK_ATRIB_NAME is the module that searches the relational linked list structure to find the ith attribute name in the list. The input to the module is an integer value. This signifies the number of the attribute in question. The output of this module is the actual relation name that is found to be the ith element in the list. SEEK_ATRIB_NAME also accesses a global pointer value that points to to the relation in question. This pointer is necessary in order to access the relation name and the list of attributes, as well as to build the functional dependencies as a linked list structure under this relation.

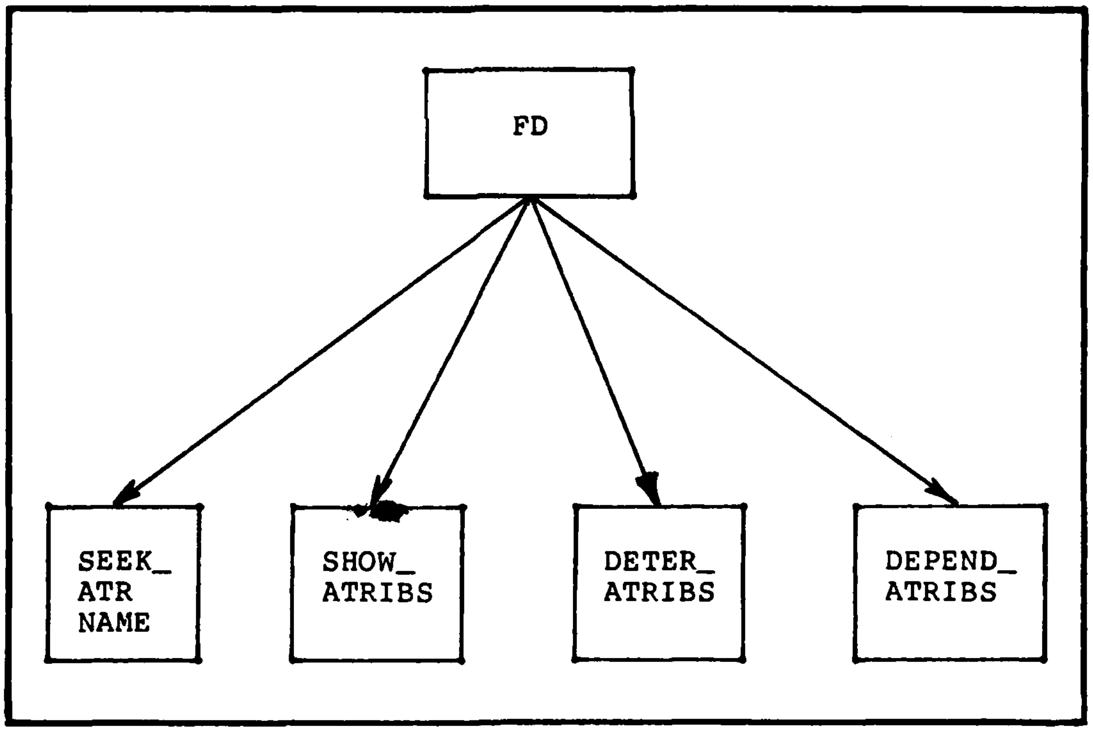


Figure 10. Strucure Chart of Module FD

The module SHOW_ATRIBS displays on the screen the attributes of the relation that needs functional dependencies defined. As discussed in the design section of this chapter, an acceptable screen display of 4 columns of attribute names, with a maximum of 20 lines of attributes would not degrade the screen readability. Due to limits imposed by the size of the screen display (80 columns wide) and due to the desire to display the four columns of names, a decision was reached to display only the first 14 characters of each attribute name. This will allow each line of display on the screen to contain four sets of the following:

2 columns	-- attribute key number
1 column	-- decimal point
1 column	-- blank (spacing)
14 columns	-- attribute name
2 columns	-- blanks (spacing)
20 columns	-- total

Four sets of the 20 columns would make up the 80 column display. A module called SCREEN, which is found in the COMMON unit, administratively handles counting the 20 lines of display that is desired when listing the attibutes. SCREEN is controlled by and modifies the global value called LINES. LINES is used to keep count of how many lines have been displayed since LINES was last set to zero (0). When LINES is incremented to 20, it causes an automatic pause and displays a user instruction that says "Press <RETURN> to continue". This feature allows the DBA to leisurely read the entire 20 lines on the screen, then continue at will.

SUMMARY

The module FD and its submodules are responsible for user friendly interaction with the DBA, while he inputs the functional dependencies of unnormalized relations. The DBA selects from a menu of attributes those attributes that comprise the determinant attribute set and those that comprise the dependent attribute set of each functional dependency of the relation in question. Module FD also gives the DBA the options to accept or reject in part or in whole the individual functional dependencies as they are being input.

VI. NORMALIZE MODULE

INTRODUCTION

The automatic normalization of relations into Third Normal Form (3NF) will produce relations that can significantly decrease the idiosyncrasies of relational databases. By reducing unnormalized relations into 3NF realtions, the obvious problems of redundancy of data and other addition/deletion anomilies can be eliminated while maintaining the integrity of the data. This chapter covers the design and implementation of the subsystem which normalizes needed relations into 3NF.

The module NORMALIZE is designed so that it is transparent to the DBA. After the module FD has aided the DBA in defining functional dependencies, NORMALIZE will be called by the executive TNT module to reduce the relation in question to 3NF. NORMALIZE is passed a pointer value that points to the relation that has just had its functional dependencies defined. With this pointer, NORMALIZE can access the relation and perform the needed normalization.

MINIMAL SET

Based on an understanding of minimal set, as defined in Chapter II, the design of the normalization segment of this thesis effort closely follows Ullman's work. By constructing a minimal set of functional dependencies, the resultant family of dependencies is in 3NF (Ref. Ullman:193).

DESIGN

The data flow graph and structure chart for NORMALIZE are found in Figures 11 and 12, respectively. As can be deduced from examining the data flow graph of NORMALIZE in Figure 11, the main thrust of this module is sequential in nature. That is to say, one procedure follows another one and so on until the last procedure is executed. Also the output of one procedure is the direct input of the next procedure. The procedures were designed with the concept of functionality in mind. Each module performs a specific function and only that function. There are six submodules (procedures) that module NORMALIZE calls into execution. They are called HIGH_LOW, BREAKOUT, RMOVE_SUBS, RMOVE_TRANS, REGROUP, and NEW_REL.

The module HIGH_LOW is the first submodule called by NORMALIZE. HIGH_LOW's function is to sort the relation's functional dependencies in a descending manner, with respect to the number of attributes in the determinant side of the dependencies. Upon completion of HIGH_LOW, the functional dependency pointer in the relation's structure will point to a functional dependency that has the largest number of attributes in the determinant side of the dependency. Then each functional dependency, including this first dependency, will in turn point to another functional dependency with an equal or smaller number of attributes or it will be the last functional dependency and its pointer will be "NIL". The impact of sorting the

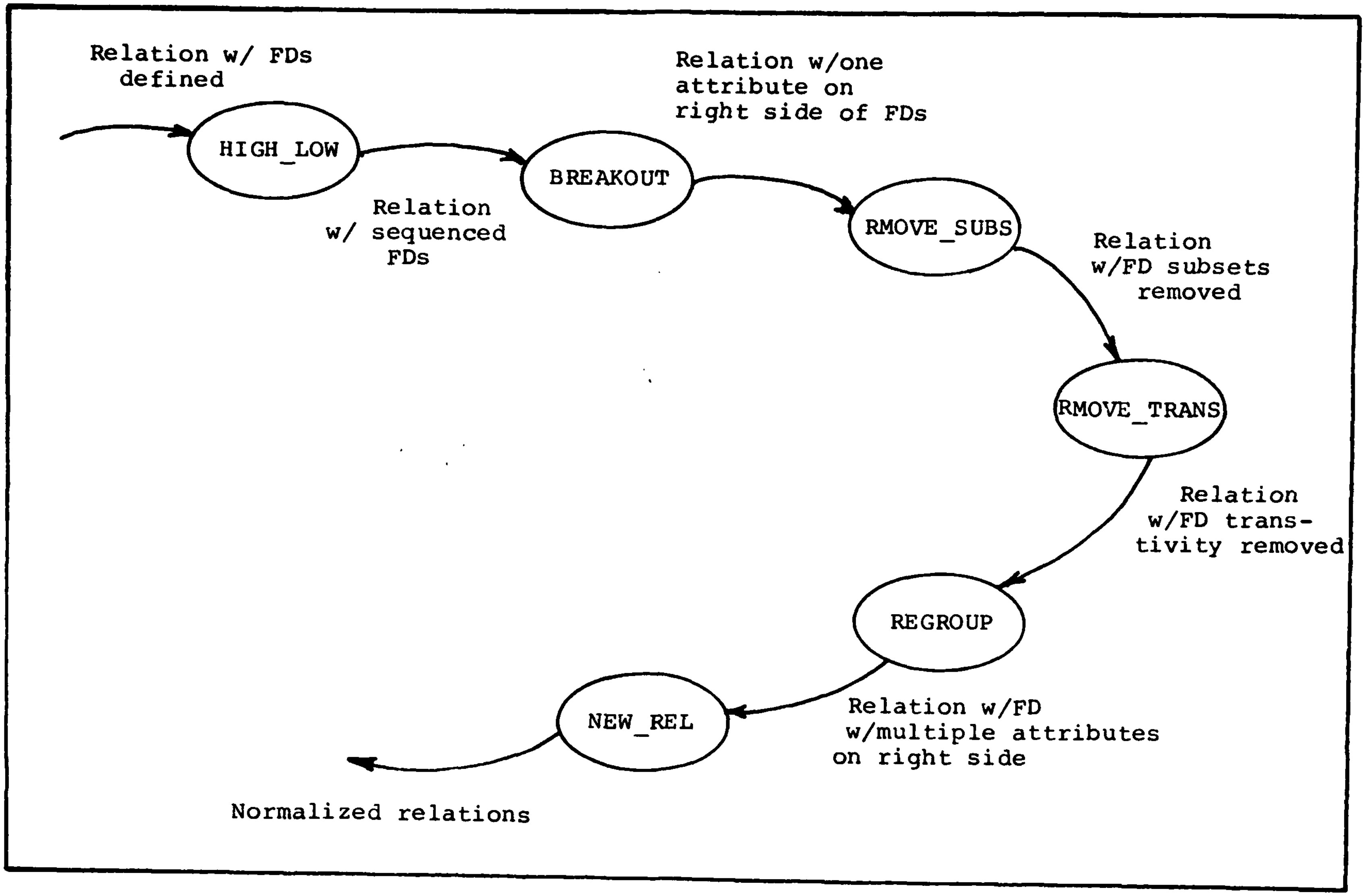

Relation w/ FDs defined
HIGH_LOW
Relation w/ sequenced FDs
BREAKOUT
Relation w/one attribute on right side of FDs
RMOVE_SUBS
Relation w/FD subsets removed
RMOVE_TRANS
Relation w/FD trans-tivity removed
REGROUP
Relation w/FD w/multiple attributes on right side
NEW_REL
Normalized relations

Figure 11. Data Flow Graph of NORMALIZE Module

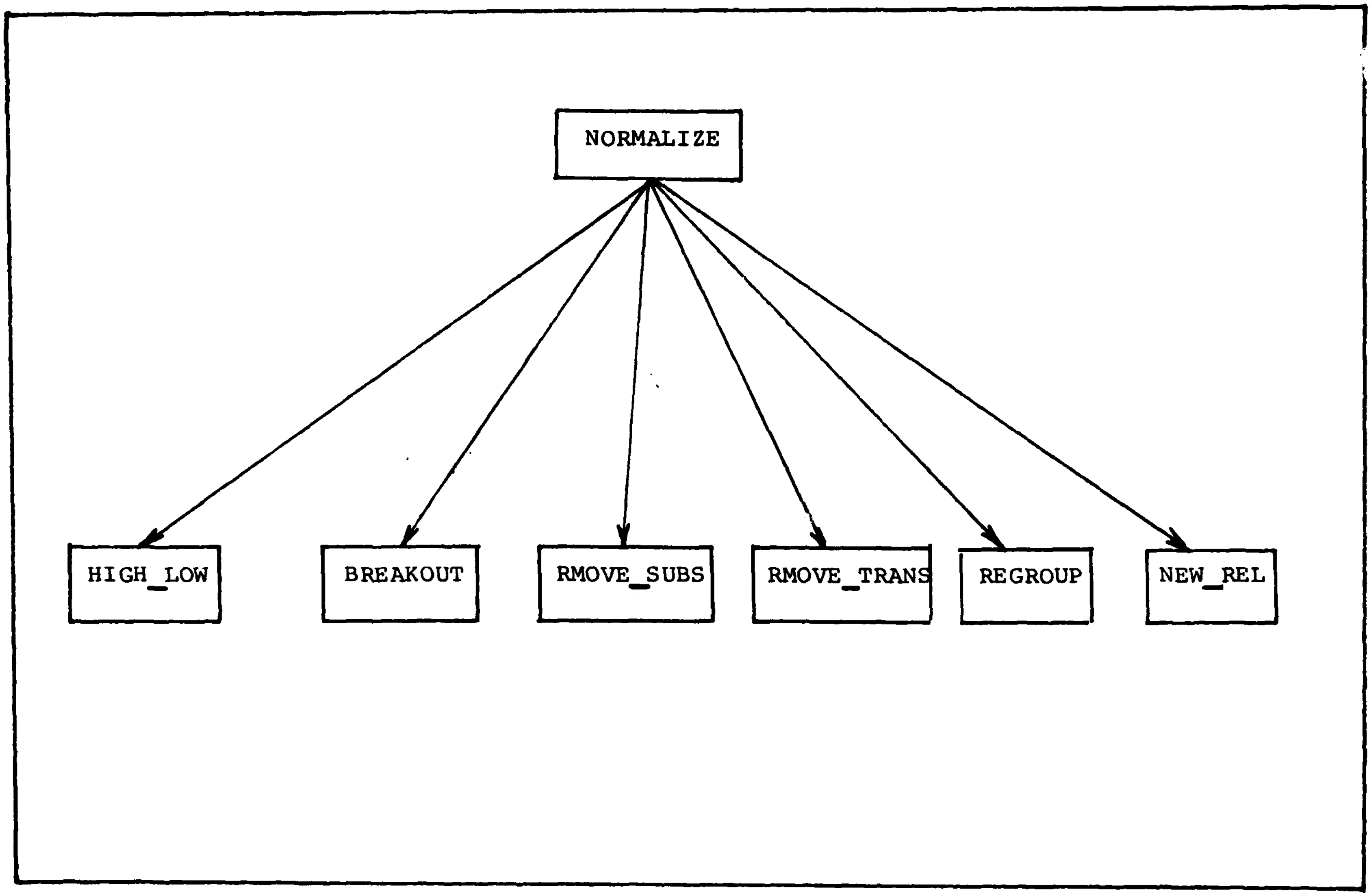


Figure 12. Structure Chart for NORMALIZE Module

functional dependencies at this particular time will become apparent later when discussing some of the other modules, but the reason for doing it now is strictly because of time efficiency.

The module BREAKOUT performs the first step in obtaining a minimal set from the family of functional dependencies. BREAKOUT takes each functional dependency that the DBA defined and separates it into functional dependencies that have the same set of determinant attributes, but with a single attribute as the dependent set. For example, if F is a family(set) of functional dependencies for a relation, which consists of the following dependencies:

```
A B --> C D E
A   --> C X
C   --> X
X   --> Z Y
```

then the results of BREAKOUT would be a set of dependencies as follows:

```
A B --> C
A B --> D
A B --> E
A   --> C
A   --> X
C   --> X
X   --> Z
X   --> Y.
```

As can be seen from the above example, all previously defined dependencies can still be obtained from the "new" set, so there is no loss of structure with in this step.

The second step in developing a minimal set is to remove dependencies that are found to be redundant in their definition. The module RMOVE_SUBS will examine each dependency in the list of functional dependencies and remove the dependency if any other functional dependency exists that has a determinant subset of the original while determining the same single attribute. For instance, by examining the above example again, it can be seen that the following dependencies:

```
A B --> C
A B --> D
A B --> E
A   --> C
A   --> X
C   --> X
X   --> Z
X   --> Y
```

can be reduced by this simple procedure. The resultant set of dependencies would be as follows:

```
A B --> D
A B --> E
A   --> C
A   --> X
C   --> X
X   --> Z
X   --> Y
```

By examination, it can be seen that the dependency "A B --> C" was eliminated. The question is "Why?". Since there were two dependencies ("A B --> C" and "A --> C") that determined the same single attribute and one of these dependencies had as a determinant set of attributes a subset of the other dependency in question, then the dependency with the subset of attributes as the determinant set of attributes was retained, while the dependency with the larger set of attributes was eliminated. In other words, since "A --> C", then there was no need to retain the dependency "A B --> C". The explanation for this procedure is simply stated in that if "A" can get you "C", then why carry along "B" if it is not necessary.

The third step in developing a minimal set of functional dependencies is the removal of all transitive dependencies. A transitive dependency exists if a dependency can be removed from the family of dependencies and the resultant set is an equivalent family. The module RMOVE_TRANS is responsible for eliminating any transitive dependencies that exist in the family of dependencies. The most recent example shows seven dependencies in the current family of functional dependencies. They are :

```
A B --> D
A B --> E
A   --> C
A   --> X
C   --> X
X   --> Z
X   --> Y
```

By careful scrutinization, it can be seen that one dependency can be eliminated by the above procedure. By examining the value on the right hand siαe of each dependency, the module searches to see if there are any dependencies with single attributes in the determinant side that are the same as the value in question. By beginning at the top of the list, the module looks to see if "D" (right hand value) can be found as a single attribute in the determinant side of any other dependencies. It is not, so the module examines the next dependency in question to see if its right hand side fits the above description. The value "E" also cannot be found as a single determinant attribute in a dependency. As RMOVE_TRANS progresses down the list, it becomes apparent that the third dependency "A --> C" fits what is being searched for. The value "C" of the third dependency is also found on the left hand side of the fifth dependency, "C --> X".

With this realization, it can be deduced that since "A -- > C" and "C --> X", that "A" implicitly determines "X". At this point, it is desired to see if there is an explicitly defined dependency "A --> X". In this example the dependency "A --> X" does exist, so it should be removed. This procedure should continue until the dependency list has been exhausted and any existing transitive dependencies have been removed. In this example, there is only one transitive dependency, so the remainder of the dependencies are necessary and meaningful. The resultant set of dependencies would be as follows:

```
A B --> D \
A B --> E  \
A   --> C   \  Minimal
C   --> X   /    Set
X   --> Z  /
X   --> Y /
```

At this time, the construction of a minimal set of functional dependencies is complete. In addition, each dependency, if treated as a relation itself, would satisfy the definition of a relation in 3NF (Ref. Ullman:194). Since the dependencies are just dependencies, there are two more steps required before they can be evolved into relations. The first step will be performed by REGROUP and the second step will be handled by NEW_REL.

The module REGROUP is designed to examine the resultant minimal set and collect functional dependencies with like determinant attribute sets into groups. If this function was performed on the previously defined minimal set, then the following dependencies would result:

```
A B --> D E  \
A   --> C     \  Regrouped
C   --> X     /     Set
X   --> Z Y  /
```

As can be seen, the six dependencies in the minimal set have now been reduced to four dependencies.

The question arises of why should the dependencies be grouped back together? Since relational databases have as a prime characteristic the elimination of redundancy of data, it would seem wasteful to create two (or more) relations with the same determinant attributes. If the family of dependencies were examined prior to the regrouping step, it can be seen that there are six dependencies with a total of 14 attributes. If each dependency evolved into a relation itself, then there would be six relations with these 14 attributes. It is easy to see that there is redundancy because of the duplication of determinant attributes. By looking at the resultant family of dependencies after they were regrouped, it can be seen that the four dependencies with 11 total attributes is more effective. This amounts to a savings of 21 per cent of required memory in this small example. In larger databases, there could be a significantly greater savings.

The only possible objection to this procedure would be in the area of security. For instance, in the previous example, if the DBA gave authorization to a user to access the attribute "D", but did not want him to access the attribute "E", then there would be problems. Since both attributes are in the same dependency the user could access both of them. Because this objection is beyond the scope of this thesis, a recommendation will be made for futher examination at a later date.

The second step in transforming the minimal set into useable relations is performed by NEW_REL. NEW_REL takes each of the regrouped dependencies and creates a new relation with the same attributes as the dependency. Each attribute of the new relation will have the same domain as was in the original relation defined by the DBA. NEW_REL will query the DBA to supply a name for the new relation as each relation is created. The results of this procedure on the regrouped set of dependencies would look as follows:

```
RELATION NAME : XXXXXXX
  ATTRIBUTES  : A B D E
  KEY         : A B

RELATION NAME : YYYYYYY
  ATTRIBUTES  : A C
  KEY         : A

RELATION NAME : ZZZZZZZ
  ATTRIBUTES  : C X
  KEY         : C


RELATION NAME : UUUUUUUU
```

```
ATTRIBUTES   : X Z Y
KEY          : X
```

The relation names would have to supplied by the DBA at the time they were created.

The result of NORMALIZE is the transformation of unnormalized relations into relations that are in 3NF. In this example, it has been shown that a relation with four DBA supplied functional dependencies could be normalized into four separate relations that have had both redundancy of data and transitive dependencies removed. It is entirely coincidential that there were four dependencies originally defined and four resultant relations. There is no way to predict the outcome of this procedure because if the dependencies were in a different order, a different minimal set would have been probable. Every other step would have been affected also.

<u>NORMALIZE IMPLEMENTATION</u>

The module NORMALIZE is used as a calling executive for the six submodules that comprise this segment of the system. This allows the modules to be broken up into segments according to their function. The six submodules which will be called are HIGH_LOW, BREAKOUT, RMOVE_SUBS, RMOVE_TRANS, REGROUP, and NEW_REL. They are listed in the order in which they will be called. This is also the order that they will be discussed.

The module NORMALIZE will reference the COMMON pointer value that points to the relation that requires normalization. By using this method, there is not a requirement to pass any values when NORMALIZED is called.

<u>Module HIGH_LOW</u>

The module HIGH_LOW performs a simple sort on the functional dependencies of the relation in question. In order to reference the relation, HIGH_LOW will use the pointer value that NORMALIZE also uses.

<u>Module BREAKOUT</u>

The module BREAKOUT performs its function by accessing the linked list of sorted functional dependencies and then creating additional dependencies by separating the original dependencies into as many dependencies as there are attributes in the dependent attribute set.

BREAKOUT will begin execution by examining the first dependency, leaving the first dependent attribute untouched in the functional dependency, then creating new record structures and inserting attribute values for the rest of the dependent attributes. BREAKOUT will continue down the linked list until all functional dependencies have been broken into dependencies with single attributes in the dependent attribute set.

<u>Module RMOVE_SUBS</u>

This module is designed to remove dependencies which have subsets of determinant attributes as determinant attributes in other realtions. Each dependency also has to have the same dependent attribute on the right hand side of the dependency in order to be a candidate for removal.

RMOVE_SUBS begins by accessing the pointer to the relation, which has a pointer to the linked list of the relation's functional dependencies. This module examines the dependent attribute of the first dependency, then it uses AUXPTR to search the list to see if there is another dependency with the same dependent attribute. If a dependency is not found, then the MAINPTR is advanced to the next dependency and the process begins all over again. If there is a dependency found that has the same single dependent attribute set, then there is a check to see if the determinant set is a subset of the original relation in question. If not, AUXPTR is advanced while trying to locate another dependency with a similar dependent attribute set.

If both the dependent attribute sets are the same and if the dependency at the AUXPTR is a subset of the dependency at the MAINPTR, then the main dependency is a candidate for deletion. This is performed by manipulating pointers which effectively removes the the functional dependency from the list.

Once the MAINPTR has traversed the entire linked list, RMOVE_SUBS has completed execution. Control is passed back to NORMALIZE so RMOVE_TRANS can be called.

Module RMOVE_TRANS

This module begins at the head of the linked list of functional dependencies and searches for transitive dependencies within the family of functional dependencies.

This procedure is performed by manipulating a set of three pointers. The required pointers are the MAINPTR, the SNGLPTR, and the AUXPTR. The MAINPTR will be the pointer which traverses the linked list of functional dependencies searching for the first dependency of a transitive dependency situation. MAINPTR will be advanced to the next functional dependency after no transitive dependencies are found or after there are no more transitive dependencies to be removed.

SNGLPTR will point to the first dependency in the list that has a single attribute in the determinant attribute set. This pointer will be a constant pointer for each relation to be normalized. Since there is a need to access the single determinant dependencies within the set of functional dependencies, SNGLPTR provides an immediate access method to the first dependency of this type.

AUXPTR will point to dependencies as transtivity is trying to be established. In fact, at the beginning of each separate search for transtivity for each particular relation, AUXPTR will be set back to equal SNGPTR. The reason for this is that since each dependent attribute set is a single attribute, all multiple determinant attribute sets can be bypassed and just the single determinant dependencies need be examined. Looking back at the example of the family of dependencies before RMOVE_TRANS was executed, the following dependencies can be recalled:

```
                   MAINPTR    SNGLPTR    AUXPTR
A B --> D            <--
A B --> E
A   --> C                       <--        <--
A   --> X
C   --> X
X   --> Z
X   --> Y
```

The original position of the pointers are shown in the above example. SNGLPTR will always remain at the first occurance of a single determinant attribute set. The MAINPTR will traverse the list attempting to find the beginning of a transitive dependency. Once MAINPTR has reached the end of the linked list, then the submodule RMOVE_TRANS has completed execution and control is passed back to NORMALIZE so REGROUP can be called.

The AUXPTR will always be reset back to equal SNGLPTR when MAINPTR is advanced one position down the list. AUXPTR is used to traverse the segment of the list that contains single determinant attribute sets. The reason for this is that since there are only single attributes in the dependent attribute set, all multiple determinant attribute sets can eliminated from consideration. This will significantly reduce search time for possible transitive dependencies.

<u>Module</u> <u>REGROUP</u>

This module is responsible for consolidating into a single functional dependency all those dependencies which have similar determinant attribute sets. This consolidation is instrumental in upholding one of the main purposes of relational databases: eliminating redundancy of data. Two pointers are required to perform the needed manipulation in this procedure. There is a MAINPTR and an AUXPTR that traverse the list of dependencies seeking those that are similar in the determinant attribute set.

Module NEW_REL

Module NEW_REL will create new relations from the regrouped functional dependencies provided by the module REGROUP. NEW_REL is required to create the record structure to hold the relation definition, query the DBA for the new relation's name, and initialize all values in the relation definition. The initialization will be accomplished by accessing information that was in the original relation definition (i.e., domains,etc), as well as setting the NORMALIZE switch of this new relation to "on", so it will not be normalized again. NEW_REL will also make a check for duplicate relation names as the DBA is queried to input the new relation's name.

SUMMARY

The module NORMALIZE is the section of this thesis effort that normalizes relations into 3NF. To do this, NORMALIZE calls six submodules, which first sorts the dependencies by size of determinant sets, performs a minimal set on the dependencies, then regroups the dependencies, and then creates new relations from these dependencies. Control is passed back to TNT when NORMALIZE has completed execution.

VI. TESTING, CONCLUSIONS, and RECOMMENDATIONS

TESTING

The implemented portion of this thesis effort was presented to students of the present database class here at AFIT. After executing the compiled system, each was able to describe functional dependencies from the list of the relation's attributes diplayed. None noted any substantial difficulties with the process. Some of their recommendations included not allowing duplicate attribute selections within the same relation, as well as including some additional screen houskeeping segments. These are legitimate recommendations and might be good choices for future implementation.

CONCLUSIONS

This thesis effort had as an original goal the design an implementation of an automated system to normalize relations in a relational database. This entailed integration with one of two existing databases within the Digital Engineering Lab (DEL) at the Air Force Institute of Technology (AFIT). The existing databases were the Roth Database System and the Fonden Database System. Due to its stage of completion, the Roth System was chosen because it was thought to have the best potential in aiding this effort.

This effort was implemented using UCSD PASCAL on the LSI-11s in the DEL. UCSD PASCAL proved to be an adequete programming language for accomplishing this work, but problems were encountered while using the PASCAL FILER system. The problems centered around the FILER "losing" or inadvertently writing over files already stored on disk files. This created tremendous annoyances.

This thesis effort was divided into two main areas with each area involving research, design, implementation, and testing. The two areas were the interactive user segment, which queried the DBA for functional dependencies; and the normalization segment, which normalized relations into 3NF. But, even before the systems could be advanced past a crude design stage, an in- depth analysis of the Roth System was accomplished and then modifications were made to this system so it could produce a data file which would be acceptable for this thesis effort. The needed data file was crucial from the onset of this effort. Once the data file could be created and accessed as required, this thesis effort was able to progress in a more orderly fashion.

The design and implementation of the interactive user segment of this effort was accomplished and it provides a method for the DBA to input functional dependencies of relations that require normalization. This was achieved by informing the DBA of the relation to be normalized, displaying the attributes, and then querying the DBA for the functional dependencies. This segment attempts to guide the DBA through this defining portion while inflicting as little grief as possible. The functional dependencies are stored in memory as they are described. They are later used in the normalization process. A future effort might possibly implement a system to write these dependencies to a disk file so the DBA and system users can maintain records of the manipulated relations.

The normalization segment was designed through the data flow diagram, the structure chart, and the program design language (PDL) levels. These are found in Appendix D. When implemented, the design will accomplish normalization of relations up through Third Normal Form (3NF). This particular effort is a prime candidate for further examination and completion, not only through 3NF, but all the way through 5NF. This would be an excellent thesis effort for someone interested in databases, because normalizing relations beyond 3NF can produce relations which are not desirable. The challenge will be in determining which relations are desirable as well as being worthwhile. Before attempting this effort, a detailed systems requirements and specifications analysis should be performed.

RECOMMENDATIONS

This thesis effort opens up several areas for further investigation. As mentioned before, there exists the need to develop a method to document the changes which relations experience during normalization. This would give the DBA and users access to what actually occurred through the execution of this system.

Areas to examined in the normalization segment would include, but would not be limited to, the implementation of of the designed, but unimplemented portion of this thesis effort, as well as investigating the design and implementation of segments to normalize up to 5NF. Also, a topic yet to be introduced would be the effect of such a system on the limited memory capability of the DEL's LSI-11s.

BIBLIOGRAPHY

1. Worden, R., "Relational Databases on Minicomputers," Proceedings of the Minicomputer Forum 1978, 63-77, London, England (November 1978).

2. Date, C. J., An Introduction to Database Systems, Reading: Addison-Wesley Publishing Company, 1981.

3. Hubbard, George U., "A Technique for Automated Logical Database Design," New York University Symposium on Database Design, 1978.

4. Ullman, J. D., Database Systems, Rockville: Computer Science Press, Inc., 1980.

5. Roth, Mark A., "The Design and Implementation of a Pedagogical Relational Database System," (Thesis) School of Engineering, Air Force Institute of Technology, Wright Patterson AFB, Ohio, 1979.

6. Rodgers, Linda M., "The Continued Design and Implementation of a Relational Database System," (Thesis) School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio, 1982.

7. Galitz, Wilbert O., Handbook of Screen Format Design, Wellesley, Massachusetts: GED Information Systems, Inc., 1981.

APPENDIX A

USER'S GUIDE: THE AFIT RELATIONAL DATABASE SYSTEM

THE RELATION NORMALIZATION FACILITY

INTRODUCTION AND OVERVIEW * SECTION 1

The Relation Normalization Facility described in this document is a program intended to run on a stand-alone minicomputer under the control of UCSD Pascal Operating System, Version II. All software is written in Pascal.

This system is designed to be used primarily with a CRT terminal as a CONSOLE device. The program does require some kind of fast mass storage such as a floppy disk system or better.

The function of the Relation Normalization Facility is to provide the Database Administrator (DBA) with the ability to normalize relations to Third Normal Form (3NF). This system interacts with a modified version of Lt Rodger's thesis effort concerning the AFIT Relational Database System. The modifications to Lt Rodger's effort entailed the addition of a "normalization" switch to each relation record and the appropriate code to the modules that defined relations, read relations from disk, and wrote relations to disk.

1.1 The Relation Normalization Facility: An Overview

The structure of the relation normalization facility is depicted in Figure A-1. The diagram depicts the upper level structure chart of the facility. The EXEC queries the DBA for input and output file names, as well as calls the modules SETUP, FD, NORMALIZE, and QUIT when appropriate to do so.

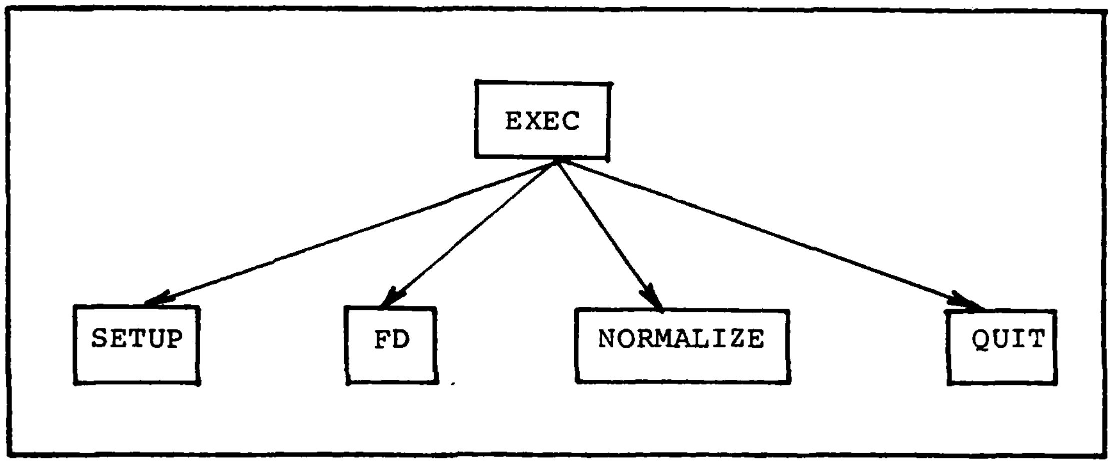


Figure A-1. Relation Normalization Facility

HOW TO USE THE RELATION NORMALIZATION FACILITY * SECTION 2

2.1 Starting the System

To start the system, execute the TNT file RUN.CODE. The disk which contains this file must remain on-line during the execution of this system in order to permit segment swapping.

2.2 Naming Input and Output Files

The system requires the DBA to first key in the disk drive number and the file name of the input file. The DBA is given a chance to verify his input. If the file does not exist on the disk indicated, an error will result and the operating system will automatically be re-initialized. The input file must be in the format of the Modified-Roth SETUP.DATA as described in Appendix C of this document. This input file can be created by executing the Modified Roth RUN.CODE file. The first step to be accomplished is to perform an "INITIALIZE." This will wipe out all previously defined domain and relation definitions. Then domains and relations should be defined as desired. A point to remember is that domain(s) of a particular relation have to be defined before the relation itself.

The Output file will also have to be named by the DBA. Its name must be different than the input file name. The DBA will be given the chance to immediately verify the name submitted as the output file name and resubmit the name if necessary. The DBA is queried by the system to submit the output file name. The query will look similar to the following:

> "Type in the disk drive number and the file
> name of the OUTPUT file that you will be using.
> - - - -"

Once the DBA has responded, the verification query will look similar to the following:

> "Is (output file name) correct? (Y/N)"

Any answer other than "Y" for "Yes" will cause the initial response to be ignored, the original output file query will then reappear, the DBA will have to respond, and the verification will have to be reaccomplished. The naming and renaming of the output file will continue until an acceptable name has been submitted and the verification response is "Y".

Once the DBA has submitted the input and output file names the system will then look for the input file on the named disk drive and if a file with the same named file is not found, the Operating System will automatically re-initialize and all previously accomplished work in the session will be obliterated.

After the named file has been found, the domain definitions will be read first and immediately written to the output file. Then the relation definitions will be read from the input file and stored in memory in the form of a linked list structure.

Once the relation definitions have been stored in memory, the first unnormalized relation, if any exist, will go through the process of defining functional dependencies and then through the normalization process. Each successively located unnormalized relation will be processed in a similar manner. The remainder of this User's Manual will discuss the two processes of defining the functional dependencies and of normalizing the relation.

DEFINING FUNCTIONAL DEPENDENCIES

When a relation has been located that requires normalization, the EXEC module first calls the module

FD to query the DBA for functional dependencies (FDs). After all the FDs have been defined for the relation in question, the EXEC will call the module NORMALIZE to reduce the relation to 3NF. Then the EXEC will look to see if any other relations in memory are in unnormalized form, so they may have their FDs defined and then be put into 3NF.

The first step in defining FDs for a relation is to define the determinant attribute set. But even before defining any attribute sets the system will display a message as follows:

"Relation xxxx is selected to be normalized," where xxx is the name of the relation. This allows the DBA to know the exact relation that is being manipulated.

Then a message is displayed to instruct the DBA that now is the time to select determinant attributes. Next is a sequence of steps that entail listing all the attributes of the relation, asking the DBA to choose one as a determinant attribute, and then asking if there are more determinant attributes to be chosen.

The listing of attributes is accomplished by displaying up to a maximum of four (4) attributes on each line of display, with 20 lines of display being the maximum allowed per screen. If more attributes exist than were displayed, the DBA is required to press the ⟨RETURN⟩ key to signal a continuation of the listing process.

Each attribute is displayed with a preceding unique key number which allows the DBA to be able to reference each attribute without having to type in the name, but just the attribute's key number. This should help avoid spelling errors and other miscellaneous mistakes.

The DBA is also asked if any more determinant attributes are to be named. If a negative response is input, then the system proceeds with the naming of dependent attributes. If an affirmative response is input, then the attributes are displayed again, the DBA is asked to input a key number to represent the attribute selected, and then the question of if there are more determinant attributes to be named is posed to the DBA.

Once all determinant attributes have been named, dependent attributes will be named in a similar manner. There is an introductory message displayed, a listing of attributes, a query for the naming of an attribute, and then the request for if there are more attributes to be named.

After the last dependent attribute of the FD is selected, a query is displayed to determine if there are more FDs to be defined. If a negative response is returned, the EXEC calls the NORMALIZE module to normalize the relation with respect to the FDs defined for the relation. If an affirmative response is returned, then the entire process of defining determinant and dependent attribute sets is repeated until there are no more FDs to be defined.

NORMALIZING RELATIONS

The normalization process when implemented will be transparent to the user. This is only because there is no user interface in this segment of the system.

APPENDIX B

INTERACTIVE AUTOMATED SYSTEM for NORMALIZATION of RELATIONS

INTRODUCTION

Data base technology has been described as "one of the most rapidly growing areas of computer and information science" (Ref 1:63). As a field, it is still relatively young; manufacturers did not begin to offer data base management products until well into the 1960's. A data base management system (DBMS) can be thought of as a system comprised of a collection of data and a set of application programs which are designed to manipulate the data. An important concept of database is that the data must be stored in the computer on direct-access devices (such as disks) in order for the computer's central processing unit to be able to utilize the data's cross-references within a reasonable amount of time. In the mid 1970's a different type of data base appeared on the scene. Its name, relational data base, implied that certain data could be viewed as a relation to other data. These relations are described in a two dimensional table consisting of horizontal rows and vertical columns.

In the Digital Engineering Laboratory (DEL) at the Air Force Institute of Technology (AFIT), the need and desire existed for a relational DBMS system, to be used as a pedagogical tool for instructing students in the design, manipulation, and use of database systems. In 1979, 2LT Mark Roth designed and partially implemented a system (Ref 2) which has served as a starting point for the continued development of the AFIT Relational Database System. In addition to the work completed by Roth, 2LT Linda M. Rodgers (Ref 3) continued the implementation of the system in 1982. An additional desire from the DEL was the design and implementation of a stand-alone system to query the Database Administration (DBA) for functional dependencies of existing unnormalized relations and the subsequent normalization of said relations. This thesis effort is the result of that desire.

Relations in a relational database can be depicted as a two dimensional table consisting of horizontal rows amd vertical columns. For example, the relation FIRST, Figure 1, is depicted by a table with 12 rows (tuples) and 5 columns (attributes). This paper will use the terms "row" and "tuple" interchangable, as well as the terms "column" and "attribute". Each tuple refers to a separate entity, while each column has values that were obtained from the domain of each attribute. In the case of "SNUM", its domain would contain "S1", "S2", "S3", and "S4". The domain of the attribute "CITY" would contain "LONDON" and "PARIS". So, each attribute has a distinct domain or value set from which its values are drawn.

SNUM	STATUS	CITY	PNUM	QTY
S1	20	London	P1	300
S1	20	London	P2	200
S1	20	London	P3	400
S1	20	London	P4	200
S1	20	London	P5	100
S1	20	London	P6	100
S2	10	Paris	P1	300
S2	10	Paris	P2	400
S3	10	Paris	P2	200
S4	20	London	P2	200
S4	20	London	P4	300
S4	20	London	P5	400

FIRST

Figure 1. Sample of Relation FIRST (Ref 4:240)

The relation FIRST has some inherent problems within its structure. The predominant problem is redundancy. The same values can be found for STATUS and CITY for all like values of SNUM. For instance, with a SNUM of S1, STATUS is always "20" and CITY is always "LONDON". This redundancy of data is expensive in terms of the additional storagerequired to hold repetitous data and in terms of the number of updates required if a supplier might move from one city to another.

At this point the term functional dependency (FD) becomes signigicant. Functional dependency is described as follows: "given a relation R, attribute Y of R is functionally dependent on attribute X of R, if and only if, each X-value in R has associated with it precisely one Y-value in relation R (at any one time)" (Ref 4:240). Functional dependencies might be thought of as a special type of integrity constraint. This means each value of the attribute X in relation R will functionally determine one and only one value for attribute Y in relation R.

With the definition of FD in mind, the main thrust of this thesis is reached, that is,normal forms and the normalization of relations. There have been described six normal forms. They are First (1NF), Second (2NF), Third (3NF), Boyce-Codd (BCNF), Fourth (4NF), and Fifth (5NF) normal form. The first three forms are stepping stones to BCNF and beyound. The "normalization process" reduces relations to the successibe six normal forms, if the relation is not already optimal. Using the pre-defined FDs, this normalization process produces a collection of new relations that are equivalent to the original relation, but from a data point of view, more desirable.. They are more desirable due to the elimination of redundancy and the creation of compact and meaningful relations.

STATEMENT OF PROBLEM

The purpose of this paper was to solve two problems. First, a system was to be designed and implemented to query the DBA for functional dependencies of relations that exist in 1NF. This segment of the thesis effort was to be a "user friendly" interactive system which would communicate with the database administrator by providing prompts, guidance, accurate display of results, and the ability to negate previous requests.

Secondly, a system was desired to normalize relations to Third Normal Form (3NF). When both goals are combined, the final system will provide the DBA a method to interactively communicate to direct the normalization of unnormalized relations.

SCOPE

The scope of this paper was to design and implement an interactive system to normalize relations. The design phase will utilize current Top Down Structured Programming (TDSP) techniques. The main effort was on the design and implementation of an algorithm to normalize relations that were defined by the Data Definition Language (DDL) (found in Appendix 2) of the Roth database system (Ref 2). In addition, a "user friendly" interface was designed and implemented to allow the user exclusive control in defining the functions dependencies of the relations to be normalized.

APPROACH

The first step consisted of an extensive literature search to examine the research already completed in the normalization area. A multitude of data base experts have written on the normalization concept. The literature search was then directed specifically to the normalization algorithm area to determine if any research had been accomplished and recorded. Very little work has been recorded in this area. Both Hubbard (Ref 5) and Date (Ref 4) have outlines for algorithms, but do not provide any detailed information towards an algorithm.

Continued research revealed that Ullman (Ref 6: 178) proposed the concept of a "minimal set" and then went on to prove that a minimal set of functional dependencies was in 3NF. With this algorithm in mind, the required modules were designed using TDSP methods and then they were compiled and validated, first as stand alone modules and then as an integrated portion of the existing Roth Database System.

SEQUENCE OF PRESENTATION

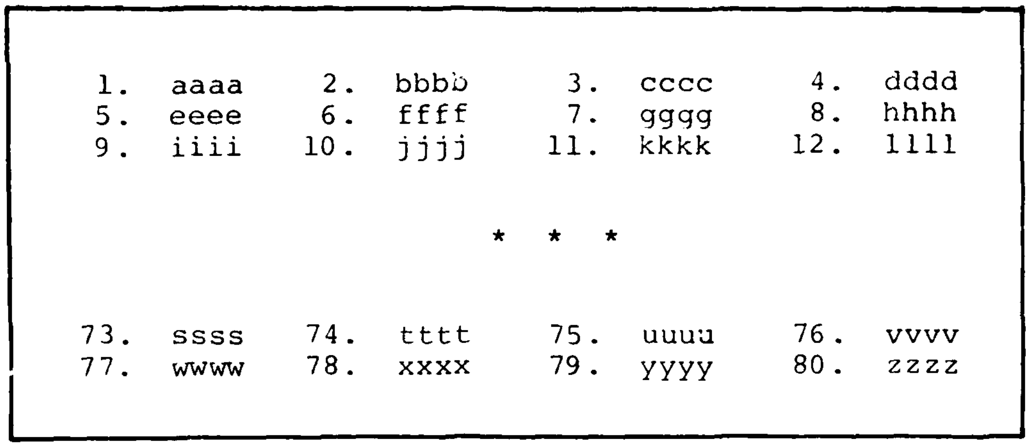
The remainder of this paper will discuss the user friendly system to define functional dependencies, the method used to derive a "minimal set," and the resultant conclusions of this effort.

DEFINING FUNCTIONAL DEPENDENCIES

The executive module TNT is designed to traverse a linked list structure of relations in memory searching for unnormalized relations. When such a relation is found, TNT calls the module FD so that functional dependencies can be input by the Database Administrator (DBA). The module FD is passed a value that points to a relation that needs to have its functional dependencies defined.

FD is designed to ask the DBA to input the attributes which comprise the determinant attribute set and the attributes that comprise the dependent attribute set of individual functional dependencies. To aid the DBA in knowing exactly which attributes are in the relation in question, a menu display of the attributes was determined to be useful. Each attribute is displayed with a key number in front of the attribute. The key numbers run sequentially from 1 to N (with N being the number of attributes in the relation). The attributes are numbered in a left to right fashion with attributes 1 to 4 on the first line, attributes 5 to 8 on the second line, and so on until all attributes or 20 lines have been displayed. If there exist more attributes than have been displayed, the DBA signals for a continuation of display by pressing the <RETURN> key. Figure A-2 depicts the screen display which will be presented to the DBA so he may select attributes to define functional dependencies.

Functional dependencies, as implied earlier, consist of two sets of attributes: the determinant set and the dependent set. The determinant set consists of attributes that imply (determine) the dependent set of attributes. The module FD first asks the DBA to input individually the attributes that comprise the determinant set, and then



```
  1.  aaaa    2.  bbbb     3.  cccc     4.  dddd
  5.  eeee    6.  ffff     7.  gggg     8.  hhhh
  9.  iiii   10.  jjjj    11.  kkkk    12.  llll

                      *   *   *

 73.  ssss   74.  tttt    75.  uuuu    76.  vvvv
 77.  wwww   78.  xxxx    79.  yyyy    80.  zzzz
```

Figure A-2. Sample Screen Display of Attributes.

asks the DBA to input individually the attributes that comprise the dependent set.

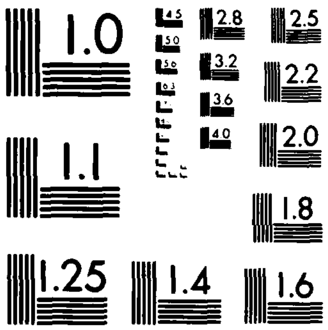
Since the module FD is the main interface with the DBA, it was desired that FD would be as "user friendly" as possible. To this end, the design of FD was accomplished with four main considerations in mind. The considerations contemplated were human factors, software, hardware, and the required applications. But even before these, there are some overall observations that should be reviewed. They are the concepts of design consistency, design standard, and design tradeoffs.

Design consistency is the first cardinal rule of all design activities (Ref 7:12). Consistency is important because it can reduce requirements for human learning by allowing skills learned in one situation to be transferred to another situation like it. While any new automated system must impose some learning requirements on its users, it should avoid burdening

AD-A127 496 INTERACTIVE AUTOMATED SYSTEM FOR NORMALIZATION OF RELATIONS(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING C T TRAVIS 18 MAR 83 2/3

UNCLASSIFIED AFIT/GCS/EE/83M-4 F/G 5/2 NL

Cont


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

productive learning with nonproductive, unnecessary activity. Inconsistencies in design are caused by differences in designers, as well as from pressure imposed by time constraints. The solution in these cases usually consists of exceptions that the user must learn to handle. People perceive a system as a single entity. To them it should look, act, and feel similar throughout. Excess learning can hinder their performance and ultimately influence their acceptance of the system.

The module FD is an interactive module that queries the DBA for functional dependencies. FD displays on the screen the name of the relation and a menu (listing) of its attributes, each with an associated key number. FD first asks the DBA to name the determinant attributes, and then FD asks the DBA for the dependent attributes. The DBA is required to input each set of attributes by keying in the associated key number of the second attribute, then the number of the second attribute, and so on until all attributes of the determinant set have been input. The querying for determinant and dependent attributes have been consistently designed. Each section of the module that requests the naming of attributes do themselves produce queries, accept responses, and display error notices similarly and consistently. Because these two sections provide similar services, consistency and procedural usage are not difficult to substantiate.

Design consistency is achieved by applying design standards. The purpose of design standards is to provide a product that is (1) consistent from both an appearance and a procedural usage standpoint, and (2) visually clear and easily used (Ref 7:32). Their objective is to reduce the subsystem processing errors and increase processing speed by faster initial learning of screen formats and discouraging "extraordinary situations" during daily activities. Some people might say that designer creativity may suffer from the imposition of design standards, but that would seem a small price to pay for an effective design.

As stated before, the module FD is consistent in its outward appearance (to the DBA) and its procedural usage. The sections that process queries, responses, and error notices are also easily used by the DBA. In fact, because of the display of a menu of attributes, the DBA is required to input only the associated number of each desired attribute, not the whole name, which would have allowed spelling errors, confusion, and frustration.

Design tradeoffs are products of: incompatible designer guidelines; time, accuracy, cost, or ease-of-use requirements; and human requirements. Design guidelines often conflict with one another or with machine processing requirements. In such conflicts the designer must weigh alternatives and reach a decision based on accuracy, time, cost, and ease-of-use requirement. This leads to another rule in user friendly systems designs: Human requirements always take precedence over machine processing requirements. While it might be easier for the designer to develop a system at the expense of the users, this must not be tolerated.

The module FD and its submodules were designed with an emphasis on ease-of-use concepts. For instance, a menu of attributes for the relation in question will be displayed with a key number in front of each attribute. When required to input attributes, the DBA will only have to input the associated key number, not the whole attribute name. The DBA also has the choice of continuing or stopping the current session each time a relation has had its functional dependencies input and the relation has been subsequently normalized. This allows the DBA the choice to curtail the current session, which causes all the relations (whether normalized or not) to be written to the new output file. At a later session the DBA might re-execute the TNT system using the last output file as the new input file. In the new session the DBA has the same choice of sequentially handling as many unnormalized relations as he wants to or as time permits. Time savings and convenience will aid the DBA in accomplishing his objectives for the session.

Human considerations are the needs and requirements of the user and are oriented toward clarity, meaningfulness, and ease-of-use. Ease-of-use has been a constant goal in the design of this thesis project, as has been discussed in previous sections. Hardware and software considerations reflect the physical constraints of the terminal on which the screen will be used and the characteristics of the controlling program. They provide a framework within which the screen design must occur. The CRT display screens used with the LSI-11s in the DEL lab are limited to an 80 column by 24 line display, so it was felt that when trying to display menus of attribute names that a maximum of four columns with a maximum of 20 lines of attribute names would not degregate the readability of the screen.

Roth originally decided to allow attribute names to be up to 132 characters in length, but it was felt necessary to truncate the attribute names to the first 14 characters in order to allow the display of four columns on the screen. This truncation is only in effect for the display of the attribute names. The names remain unchanged in memory.

If the relation has more than 80 attributes, two or more screens of display will be required. After a screen has been displayed, an automatic pause will allow the DBA to examine the list as necessary. When ready, the DBA will signal for a continuation of the listing by pressing the ⟨RETURN⟩ key. Then the next set of attributes will be displayed either until the screen is full or the last attribute has been displayed. Application considerations reflect the objectives of the system for which the screen is being designed. They are the data or information building blocks which make up a screen display (Ref 7:14). The application consideration for FD is the input of accurate functional dependencies that will aid the system to correctly normalize relations.

NORMALIZATION OF RELATIONS

The automatic normalization of relations into Third Normal Form (3NF) will produce relations that can significantly decrease the idiosyncrasies of relational databases. By reducing unnormalized relations into 3NF relations, the obvious problems of redundancy of data and other addition/deletion anomalies can be eliminated while maintaining the integrity of the data.

The module NORMALIZE is designed so that it is transparent to the DBA. After the module FD has aided the DBA in defining functional dependencies, NORMALIZE will be called by the executive TNT module to reduce the relation in question to 3NF. NORMALIZE is passed a pointer value that points to the relation that has just had its functional dependencies defined. With this pointer, NORMALIZE can access the relation and perform the needed normalization.

Due to the significant amount of time spent in the beginning of this effort towards understanding the Roth Database System and pinpointing potential modifications, this normalization segment was designed and the Program Design Language (PDL) for each module was written (can be found in Appendix D of this document), but was not implemented. Implementation should be of little effort for a team of qualified PASCAL programmers.

MINIMAL SET

The design of the normalization segment of this effort closely follows Ullman's work. By constructing a minimal set of functional dependencies, the resultant family of dependencies is in 3NF (Ref 6:133-4).

The main thrust of the module NORMALIZE is sequential in nature. That is to say, one procedure follows another one and so on until the last procedure is executed. Also, the output of one procedure is the direct input of the next procedure. The procedures are designed with the concept of functionality in mind. Each module performs a specific function and only that function. There are six submodules (procedures) that module calls into execution.

Ullman states three requirements are necessary for a set of functional dependencies (FDs) to be a "minimal set" and subsequently in 3NF. First, he says that each FD in the family of FDs can only have one dependent attribute in the dependency. This step entails creating as many new FDs as there are dependent attributes in the original FD. These new FDs will be substituted for the original FD. For example, if the FD of $A\ B \longrightarrow C\ D\ E$ exists in a family of FDs, then the result of this step would appear as the following:

$A\ B \longrightarrow C$

$A\ B \longrightarrow D$

$A\ B \longrightarrow E$

As can be seen, the original FD with three dependent attributes was replaced by three new FDs. The significance and meaning of the original FD has not been lost in this step because FDs with similar determinant attributes can later be regrouped as needed.

The second requirement for establishing a minimal set is to remove dependencies that are found to be redundant in their definition. This means to remove the FDs which have a subset of determinant attributes that determines the same set of dependent attributes. For instance, if the FDs $A\ B \longrightarrow C$ and $A \longrightarrow C$ exist, then the FD $A\ B \longrightarrow C$ should be eliminated because the FD $A \longrightarrow C$ will still exist and because if A by itself implies C, then why retain the FD where A and B implies C. This step has eliminated the redundancy of the FDs.

The third requirement is to remove all transitive dependencies within the family of FDs. A transitive dependency exists if a dependency can be removed from the family of dependencies and the resultant set is an equivalent family. If three FDs exist in a family of FDs such as: $A \longrightarrow C$, $A \longrightarrow X$, and $C \longrightarrow X$; then it can be seen that A implies X directly in one dependency as well as through the attribute C, i.e., $A \longrightarrow C$ and $C \longrightarrow X$. Because a transitivity exists, the FD $A \longrightarrow X$ should be eliminated.

At this time, the construction of a minimal set of functional dependencies is complete. In addition, each dependency, if treated as a relation itself, would satisfy the definition of a relation in 3NF (Ref 6:194).

Because relational databases strive to reduce redundancy of data, a process beyond deriving the minimal set is necessary to continue this effort of reduction of redundancy. This process is one mentioned in an earlier section; that is, the grouping together of dependencies with similar determinant attributes. This will eventually eliminate the creation of two (or more) relations with the same determinant attributes.

Once a minimal set has been derived and FDs with similar determinant attributes have been grouped as single FDs, then the last step will be to create separate relations out of each FD within the family. For instance, if the FD A B $\longrightarrow$ C existed in a family of FDs, then a relation would be created with the attributes A, B, and C. The key would be the attribute set A B.

CONCLUSION

Unnormalized relations in a relational database can be automatically reduced to 3NF if functional dependencies are provided for use by this system. This effort produced a system which interfaces with the Roth Database System and which when fully implemented can interactively query the DBA for functional dependencies of a relation and reduce the relation to an equivalent set of relations in Third Normal Form.

APPENDIX C

Program Design Language Code

for NORMALIZE Submodules

Module HIGH_LOW

```
(************************************************************
*  PURPOSE : Sort FDs of the relation that NREL points to  *
*            in a descending order, by the number of        *
*            attributes ineach FD's determinant attribute   *
*            set.                                           *
*  GLOBAL VARIABLES USED :                                  *
*     NREL - pointer to relation in question.               *
*                                                           *
*  GLOBAL VARIABLES MODIFIED : None.                        *
*                                                           *
************************************************************)

PROCEDURE HIGH_LOW

BEGIN
  USE QUICK SORT METHOD FOUND IN FUNDAMENTALS OF DATA
      STRUCTURES, HOROWITZ and  SAHNI, COMPUTER SCIENCE
      PRESS, 1976, pp 347-350.
END.
```

Module BREAKOUT

```
(************************************************************
*  PURPOSE: Insure that each functional dependency (FD)     *
*           contains only one attribute in the dependent    *
*           attribute set.  This is accomplished by tra-    *
*           versing the linked list of FDs, selecting those*
*           that contain two or more attributes in the      *
*           dependent set, and then creating "new"(equiv-   *
*           alent) FDs that have the same determinant set   *
*           of attribute(s) witha single attribute as the   *
*           dependent set (ie, the FD "A B C -- D E" would *
*           produce the FDs "A B C -- D" and "A B C -- E").*
*                                                           *
*  GLOBAL VARIABLES USED :                                  *
*     NREL - pointer to relation in question.               *
*     TFD  - functional dependency record structure.        *
*     PART - attribute record structure.                    *
*                                                           *
*  GLOBAL VARIABLES MODIFIED: None.                         *
*                                                           *
************************************************************)

PROCEDURE BREAKOUT

BEGIN
  SET TPTR1 TO NRELΛ.NEXTFD  (* head of list  *)
  DOWHILE TPTR1<> NIL
    BEGIN
      DOWHILE NUMBER DEPENDENT ATTRIBUTES> 1
        BEGIN
          CREATE NEW FD STRUCTURE
          MOVE DETERMINANT SET TO NEW STRUCTURE
          MOVE 1st DEPENDENT ATTRIBUTE TO NEW STRUCTURE
          REMOVE 1st DEPENDENT ATTRIBUTE FROM TPTR1Λ.TOPTR
               LIST
          INSERT NEW FD STRUCTURE INTO FD LIST
        ENDDO
      ADVANCE TPTR1 DOWN FD LIST ONE RECORD
    ENDDO
END.  (*  BREAKOUT   *)
```

Module RMOVE_SUBS

```
(***********************************************************
*  PURPOSE : Remove dependencies which have subsets of      *
*            determinant attribute(s) as determinant        *
*            attributes in other funtional dependencies     *
*            which determine the same single dependent      *
*            attribute.                                     *
*                                                           *
*  GLOBAL ATTRIBUTES USED :                                 *
*     NREL - pointer to relation in question.               *
*                                                           *
*  GLOBAL VARIABLES MODIFIED : None.                        *
*                                                           *
*  INTERNAL VARIABLES USED :                                *
*     MAINPTR - pointer to FD in question.                  *
*     AUXPTR  - pointer to possible subset FD.              *
*                                                           *
***********************************************************)

PROCEDURE RMOVE_SUBS

BEGIN
  SET MAINPTR TO 1st FD  (* head of list  *)
  SET AUXPTR TO MAINPTR^.NEXTFD
  DOWHILE MAINPTR^.NEXRPTR  NIL
    BEGIN
      DOWHILE AUXPTR<>NIL
        BEGIN
          IF MAINPTR^.TOPTR^.PARTNAME EQUALS
               AUXPTR^.TOPTR^.PARTNAME THEN
            IF AUXPTR DETERMINANT SET IS SUBSET OF MAINPTR
                  DETERMINANT SET THEN
              REMOVE MAINPTR FD
          ADVANCE AUXPTR ONE FD DOWN LIST
        ENDDO
      ADVANCE MAINPTR ONE FD DOWN IN LIST
      SET AUXPTR TO MAINPTR^.NEXTPTR
    ENDDO
END.
```

Module RMOVE_TRANS

```
(************************************************************
*  PURPOSE : Searches linked list of FDs to find transitive*
*            dependencies and then removes them.           *
*                                                          *
*  GLOBAL VARIABLES USED :                                 *
*     NREL - pointer to relation in question.              *
*                                                          *
*  GLOBAL VARIABLE MODIFIED : None.                        *
*                                                          *
*  INTERNAL VARIABLES USED:                                *
*     MAINPTR - pointer to possible 1st dependency of      *
*               transitive dependency.                     *
*     SNGLPTR - pointer to 1st dependency in list that has *
*               single attribute in determinant set.       *
*     AUXPTR1 - extra pointer.                             *
*     AUXPTR2 - extra pointer.                             *
*                                                          *
************************************************************)

PROCEDURE RMOVE_TRANS

BEGIN
  SET SNGLPTR TO 1st FD WITH SINGLE DETERMINANT SET
  SET MAINPTR TO 1st FD IN LIST (* head of list  *)
  DOWHILE MAINPTR^.NEXTFD   NIL
    BEGIN
      DOWHILE AUXPTR1.NEXTFD <> NIL
        BEGIN
          SET AUXPTR1 TO SNGLPTR
          SET AUXPTR2 TO SNGLPTR
          IF MAINPTR DEPENDENT ATTRIBUTE SET EQUALS
             AUXPTR1's DETERMINANT ATTRIBUTE SET THEN
            DOWHILE AUXPTR2 <> NIL
              BEGIN
                IF AUXPTR2's FD EQUALS FD MADE UP OF
                   (MAINPTR's DETERMINANT SET AND AUXPTR1's
                    DEPENDENT SET) THEN
                 REMOVE AUXPTR2's FD
                ADVANCE AUXPTR2 DOWN LIST ONE RECORD
              ENDDO
          ADVANCE AUXPTR1 DOWN LIST ONE RECORD
        ENDDO
      ADVANCE MAINPTR DOWN LIST ONE RECORD
    ENDDO
END.
```

Module REGROUP

```
(***********************************************************
*  PURPOSE : Consolidation into a single functional        *
*            dependency all dependencies that have similar *
*            determinant attribute sets.                   *
*                                                          *
*  GLOBAL VARIABLES USED :                                 *
*     NREL - pointer to relation in question.              *
*                                                          *
*  GLOBAL VARIABLES MODIFIED : None.                       *
*                                                          *
*  INTERNAL VARIABLES USED :                               *
*     MAINPTR - pointer to FD in question.                 *
*     AUXPTR  - pointer to possible FD to be joined.       *
*                                                          *
***********************************************************)

PROCEDURE REGROUP

BEGIN
  SET MAINPTR TO HEAD OF FD LIST
  SET AUXPTR TO HEAD + 1  OF FD LIST
  DOWHILE MAINPTR∧.NEXTFD   NIL
    BEGIN
      DOWHILE MAINPTR's DETERMINANT SET EQUALS
                AUXPTR's DETERMINANT SET
        BEGIN
          ADD AUXPTR's DEPENDENT ATTRIBUTE(S) TO MAINPTR's
               DEPENDENT ATTRIBUTE SET
          DISPOSE OF AUXPTR's FD  (* remove it  *)
        ENDDO
      ADVANCE MAINPTR DOWN LIST ONE FD RECORD
      SET AUXPTR TO MAINPTR
    ENDDO
END.
```

Module NEW_REL

```
(*************************************************************
*  PURPOSE : Create relations from the regrouped             *
*                functional dependencies.                    *
*                                                            *
*  GLOBAL VARIABLES USED :                                   *
*     NREL -  pointer to relation in question.               *
*                                                            *
*  GLOBAL VARIABLES MODIFIED : None.                         *
*                                                            *
*************************************************************)

PROCEDURE NEW_REL

BEGIN
  DOWHILE NRELΛ.NEXTFD   NIL
    BEGIN
      CREATE RELATION STRUCTURE
      SET NORMALIZE SWITCH TO "ON"
      DISPLAY ATTRIBUTES
      QUERY DBA FOR NEW RELATION NAME
      BUILD ATTRIBUTE LIST FROM FD's DETERMINANT AND
             DEPENDENT SETS
      LINK KEYPTR LIST FROM FD's DETERMINANT SET
      SET NEXT FD TO NIL
      SET SECURITY RECORD FROM NREL SECURITY RECORD INFO
      SET MODIT, TEMPEXIST, and  ATTACH FROM NREL's INFO
      SET FILER FROM NREL's FILER
      INSERT THIS NEW RELATION INTO RELATION LIST
         IMMEDIATELY BEHIND NREL's RELATION
      DISPOSE OF NRELΛ.NEXTFD (* remove head FD from list *)
    ENDDO
END.
```

VITA

Charles T. Travis was born July 12, 1949, in Brewton, Alabama. He graduated with a B.S. in Computer Science in 1972 from the University of Southern Mississippi. On March 18, 1983, he graduated with an M.S. in Information Systems from the Air Force Institute of Technology.

Permanent Address: 317 Farrell Street
Picayune, Mississippi 39466

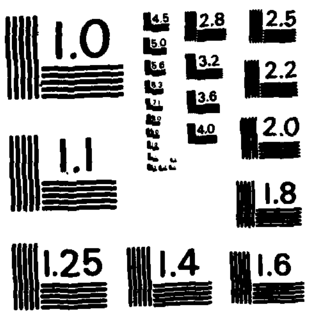
AD A127 496 INTERACTIVE AUTOMATED SYSTEM FOR NORMALIZATION OF RELATIONS(U) AIR FORCE INST OF TECH WRIGHT PATTERSON AFB OH SCHOOL OF ENGINEERING C T TRAVIS 18 MAR 83

3/3

UNCLASSIFIED AFIT/GCS/EE/83M-4 F/G 5/2 NL

SUPPLEMENTARY INFORMATION

END
DATE FILMED
2 84
DTIC


1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

SUPPLEMENTARY

INFORMATION

ERRATA

AD-A127 496

Page 86 is not missing. Document wae misnumbered per AFIT.

DTIC-DDAC

9 Dec 83

ATE

LMED

8