1985

Computer assisted relational data base logical design /

Chien-Chung J. Chuang
Lehigh University

COMPUTER ASSISTED RELATIONAL DATA BASE LOGICAL DESIGN

by

Chien-Chung J. Chuang

A Thesis

Presented in Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Industrial Engineering

Lehigh University

1984

ACKNOWLEDGMENTS

My special thanks are due to my adviser, Professor John C. Wiginton, for his advice and guidance throughout the evolution of this thesis.

I dedicate this thesis to my parents.

CERTIFICATE OF APPROVAL

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science in Industrial Engineering.

December 6, 1985
Date

John L. Wilson
Professor in Charge

S. C. Chow
Chairman
Department of Industrial Engineering

Table of Contents

ABSTRACT

This thesis presents a methodology for data base design in which the design process is organized into a series of structured and computerized procedures based on mathematical methods and implemented as an interactive computer program in NIAL (Nested Interactive Array Language).

The structured procedures are summarized below :

1. Conceptual design.

    a. Combination of local views.

    b. Determination of Elementary Relations.

2. Structure refinement.

    a. Transitivity removal.

    b. Reducing the number of Elementary Relations.

3. Design results.

The methodology can provide the following benefits :

1. Presents information helpful to the designer.
2. Performs a more thorough and consistent analysis of the design.
3. Improves the design quality.
4. Shortens the design cycle.

For the purpose of illustrating the benefits of this methodology, a prototypical case study is included.

1. INTRODUCTION

1.1 OVERVIEW

The object of this thesis is to demonstrate the feasibility of computer assisted, systematic procedures in the long and tedious processes of logical database design. The focus is on the relational logical structure. The emphasis is on automating as much as possible. The procedures used can also increase the understanding of the design concept regardless of whether or not computer assistance is used.

This thesis presents a design methodology consisting of structured procedures in logical relational database design. The algorithm for these procedures is presented and implemented in NIAL (Nested Interactive Array Language). The computer-assisted procedures are applied to a prototypical case study as an illustration of their use.

1.2 THE MODEL

The model selected for implementation is the relational structured data model.

The reasons for this are two fold :

1. The relational database model has both logical and physical characteristics. It is approximately the midpoint of the human/machine continuum, which means that data

bases designed according to the relational model need not be transformed into some other format before implementation.

2. Much more research has been done on the relational model than on any other model. Recent developments of several general purpose relational DBMSs (Data Base Management Systems) have brought the relational data base design into a new era in the field of data base design.

1.3 PREPARATIONS PRIOR TO THE COMPUTER-ASSISTED PROCEDURES

Traditionally, data base design has been a two-phase process : **logical data base design**, where the needs of people are specified ; and **physical data base design**, where logical data design is mapped into the constraints of particular program and hardware products. (See Fig 1.1)

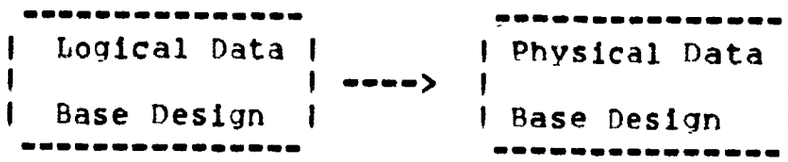


Figure 1.1

We may further divide the logical design phase into two stages : **system analysis**, where data to be stored are identified, while data names are consolidated and clarified ; and **computer-assisted procedures**, where

logical schema are developed, and the specification of data base records, their constraints and relationships are specified. (See Fig 1.2)

Figure 1.2

In order to make the greatest use of the computer-assisted procedures , before getting into these procedures, the designer should have a clear concept about the system environment, the constraints and limitation on the system design, documentation in the form of DFDs (Data Flow Diagrams), and a thorough understanding of the diverse user requirements.

2. DEFINITION OF TERMS

Before we can discuss design, it is important to know the terminology used in relational data base design.

2.1 RECORDS, ENTITY SETS, ATTRIBUTES AND DOMAINS

An **entity** may be " anything that has reality and distinctness of being in fact or in thought. ". It may be :

1. A real object, e.g. a classroom.
2. An individual, e.g. a student.
3. An abstract concept, e.g. Lehigh University.
4. An event, e.g. students taking courses.
5. A relationship, e.g. Professor instructs students [12].

A collection of occurrences of similar entities is an **entity set.** We maintain a **record** for each entity, and **record sets** for similar entities. The records refer to **attributes** of entities and contain values for these attributes, which are called **domains.**

Mealy [10] and Engles [4] have pointed out that there are three realms which we may talk about regarding information. (See Fig 2.1)

Reality	Information	Data Item
Entity Sets :	Attributes :	Domains :
Student	Student #	123, 234
	Name	J. Smith, C. Chuang
	Tel #	123-4567, 234-5678

Figure 2.1 Three Realms [8]

As can be seen in the above figure, the first realm is the real world in which there are entities and in which the entities have certain properties. The second is the realm of ideas and information existing in the minds of men and programmers. The third realm is that of data in which strings of characters or bits are used to encode items of information.

2.2 RELATIONSHIP SETS

A **relationship set** is an n-array relation on n entity sets : these entity sets need not be distinct. It is a way of representing a type of relationship between entities [12]. For example :

Suppose we have entity sets Student (E1) and Professor (E2)

E1 = [S1, S2, S3],

E2 = [P1, P2].

Then, we'll have relationship set Instruction (R) :

R = [I1, I2]

in which,

I1 = [P1, S1, S2],

I2 = [P2, S2, S3]

and is interpreted as :

PROFESSOR#1 instructs STUDENT#1, STUDENT#2.

PROFESSOR#2 instructs STUDENT#2, STUDENT#3.

2.3 KEYS

A key is a data item designated to be an entity identifier [4]. It must be associated with the property uniqueness, so that we can use it to refer to some given entity unambiguously.

2.3.1 SIMPLE KEY

A simple key is a key consisting only one data element. It is atomic and its values are unique, e.g. Student#.

2.3.2 COMPOSITE KEYS

Composite keys are keys composed of two or more data elements, all of which are required for unique identification. e.g. Student#, Course# --> Grade.

2.4 MAPPINGS

A **mapping** is a rule of association between two sets [12]. Three types of mappings are discussed below :

2.4.1 1:1 MAPPING

The mapping :

Directed-By : Department <--> Chairman

is a 1:1 mapping since every department has exactly one chairman and every chairman is in charge of one department only.

2.4.2 M:1 MAPPING

The mapping :

Advised-By : Student <<--> Professor

is a M:1 mapping since a student is advised by one professor only but a professor can advise several students.

2.4.3 M:M MAPPING

The mapping :

Taught-By : Student <<-->> Professor

is a M:M mapping since a student can be taught by several professors and a professor can teach several students.

2.4.4 FUNCTIONAL DEPENDENCY

An attribute B of a relation is functionally dependent on the attribute A of the relation if at every instant of time each A-value is associated with no more than one B-value [12]. For example :

STUDENT# <<--> LAST-NAME

is a FD, because for a given student#, there exists only one last name for it.

2.4.5 ELEMENTARY RELATIONS

A set of irreducible units which represents the transformation of the real world into a simpler model is called Elementary Relations (ERs). Elementary Relations satisfy the requirement for a single place for a single fact. They have the advantage that future steps in the design process can be based on a firm theory [12]. For

example :

The relation

STUDENT# <<--> LAST-NAME, FIRST-NAME, STATUS,ADDRESS

may be described by the following set of ERs :

ER1(STUDENT#, LAST-NAME)
ER2(STUDENT#, FIRST-NAME)
ER3(STUDENT#, STATUS)
ER4(STUDENT#, ADDRESS)

3. THE METHODOLOGY

The methodology presented here is based on the synthesis approach, that is, we commence with a set of functional dependencies(FDs) and use them to construct relations. This thesis selects concepts from James Martin [8], M. Vetter & R. N. Madison [12], and S. Atre [1], and synthesizes them into a systematic and computerized methodology.

The major concept adopted from both James Martin and S. Atre is the combination of diversified user needs (local views). The computer program simulates this process by two steps :

1. Initial Data Entry.
2. Reediting.

These two steps are iterative for the purpose of modifying the original data entry according to different user views.

The concepts discussed by M. Vetter & R. N. Madison form the central part of the methodology. These are the concept of sets and the use of mathematical methods in database design.

3.1 STRUCTURED PROCEDURES

The structured procedures developed are in three-phase : **conceptual design, structure refinement** and **design results.** The three phases are then further broken down into five steps, which again each step consists of several processes, as summarized below :

3.1.1 CONCEPTUAL DESIGN

1. Combination of local views.

    a. Data entry for simple key records.

    b. Data entry for composite key records.

    c. Generation of relation sets.

    d. Reediting.

2. Determination of Elementary Relations (ERs).

    a. Removal of the redundancy.

    b. Mapping between keys.

    c. Generation of relations for trivial dependence.

    d. Determination of ERs.

3.1.2 STRUCTURE REFINEMENT

1. Transitivity removal.

    a. Determination of transitive closure.

    b. Selecting the semantically meaningful ERs.

    c. Transitivity removal.

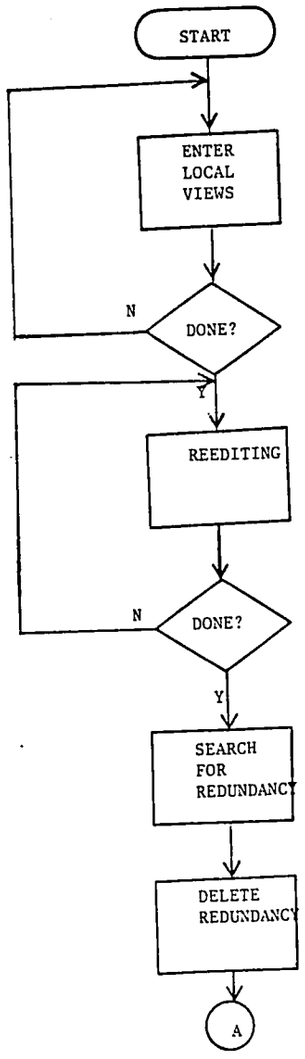
2. Reduction of the number of ERs.

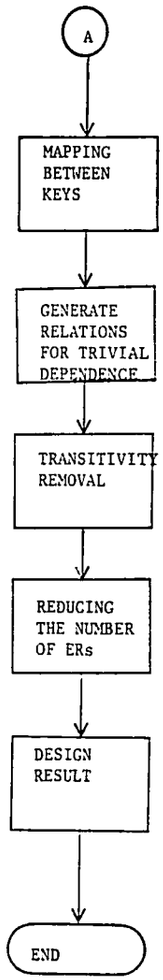
3.1.3 DESIGN RESULT

3.2 THE FLOW CHART FOR THE OVERALL DESIGN PROCESS

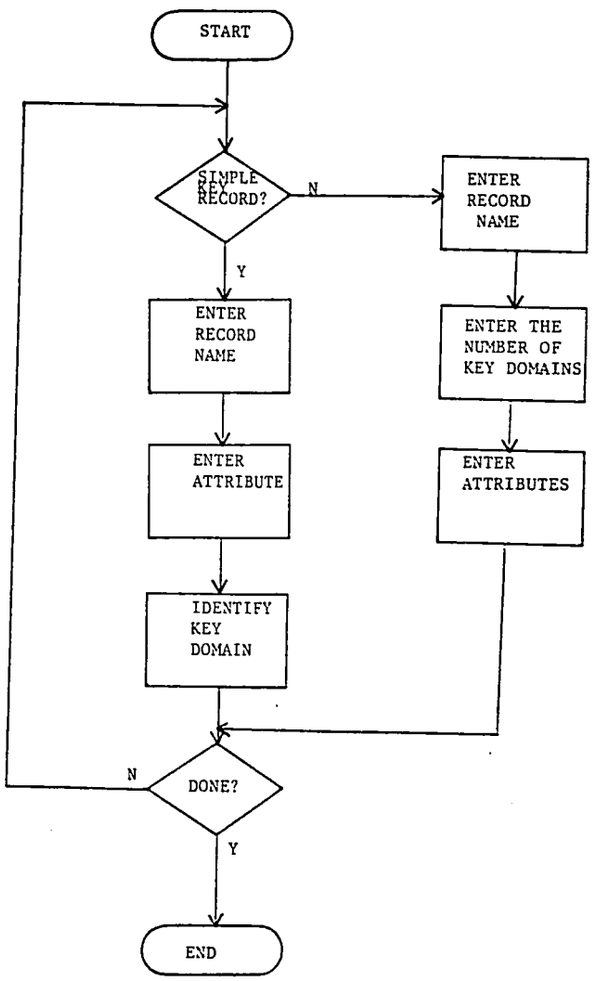
As shown in the following pages :

PROGRAM LOGIC FOR THE ENTIRE DESIGN PROCESS

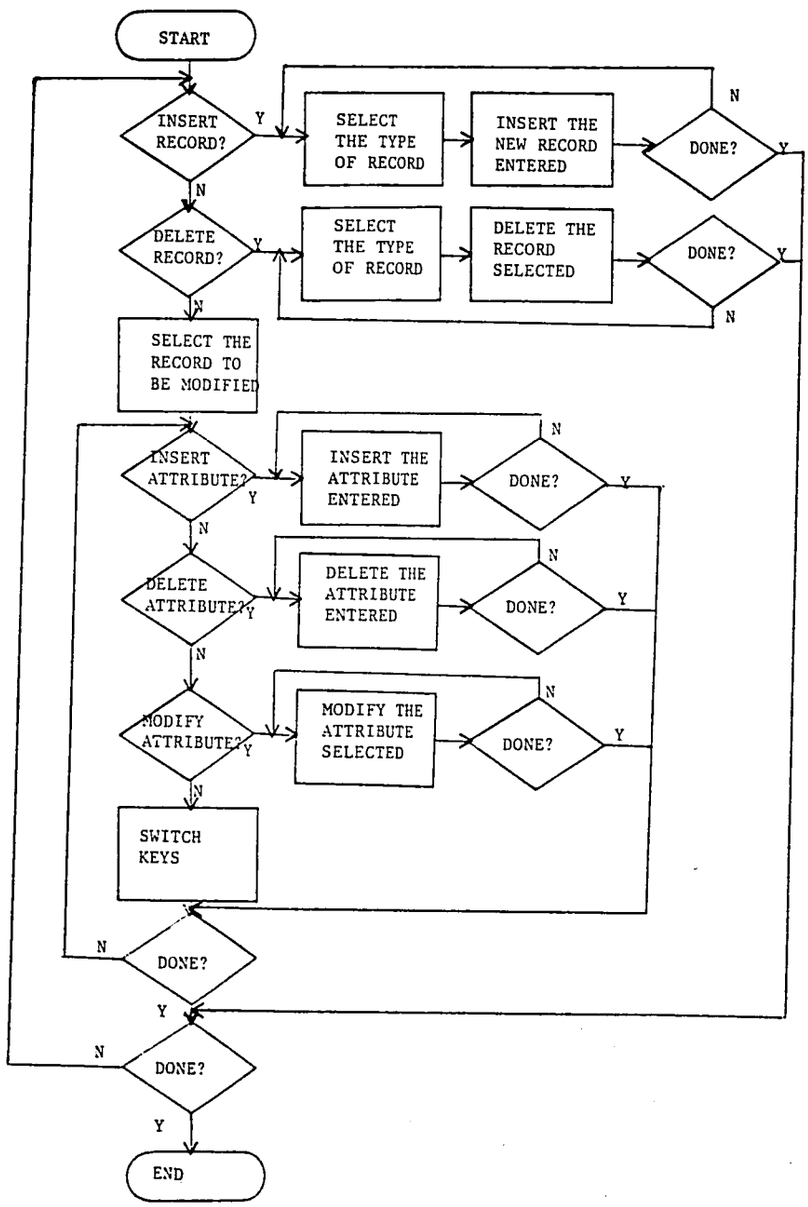



A
MAPPING BETWEEN KEYS
GENERATE RELATIONS FOR TRIVIAL DEPENDENCE
TRANSITIVITY REMOVAL
REDUCING THE NUMBER OF ERs
DESIGN RESULT
END

PROGRAM LOGIC FOR "ENTER LOCAL VIEW"



PROGRAM LOGIC FOR "REEDITING"



4. THE ALGORITHM FOR THE COMPUTER-ASSISTED PROCEDURES

4.1 TRANSITIVITY REMOVAL

Consider the two FDs :

f : A --> B

g : B --> C

a third FD - their product, can be derived :

(g o f) : A --> C

The products of FDs are transitive dependencies.

Deriving all such FDs from some initial collection of ERs yields a transitively closed collection of ERs called a **transitive closure.** It includes both derived and original ERs, some of which are redundant. An ER is redundant if it can be derived from other ERs, and the purpose here is to eliminate those redundant ERs. We can automate this procedure by using matrix operations as described below :

Suppose we have data elements :

Ai, Aj, Ak

And FDs :

Ai --> Aj and Aj --> Ak

We can derive transitive dependence :

Ai --> Ak

Which represents the composition :

Ai --> Aj --> Ak

The derived transitive dependence is then recorded in a (n x n) " connectivity matrix " M by assigning " 1 " to Mi,k.

For example :

We have data elements :

A1, A2, A3, A4

And FDs :

A1 --> A3, A3 --> A4

The connectivity matrix derived will be

	A1	A2	A3	A4
A1	0	0	1	0
A2	0	0	0	0
A3	0	0	0	1
A4	0	0	0	0

the following FD can be derived :

A1 --> A4

which represents the composition :

A1 --> A3 --> A4

the new FD is then recorded in a new matrix

	A1	A2	A3	A4
A1	0	0	0	1
A2	0	0	0	0
A3	0	0	0	0
A4	0	0	0	0

The logic presented below describes the procedures

for deriving the connectivity matrix, determining the transitive closure [12], and removing transitivity. For the algorithm of the determination of the composition for the matrix, the readers are encouraged to read reference [12].

1. Generate a list of all data elements.

2. Derive a (n x n) matrix (in which n denotes the number of data elements in the list generated in step 1) representing the relations among the data elements in the list.

    a. If there exists a relation between data elements then put " 1 " in the corresponding position of the matrix.

    b. For key attributes : put " 1 " in the corresponding position of the matrix for the mapping of the key attribute itself (i.e. employee# <--> employee#, where employee# is one of the key attributes).

    c. Otherwise, put " 0 " in the corresponding position of the matrix.

3. Determine the composition for the matrix. (As illustrated in the above example).

4. Remove the erroneous compositions derived by the situation : Ai --> Aj --> Ai.

5. Were any new composition derived ? If not, go to step 8.

6. Create a new modified matrix that includes the new composition obtained from step 3.

7. Repeat step 3, 4, 5, 6 until no new composition can be obtained.

8. Determine the new FDs derived.

9. Select, from among the FDs, semantically meaningful ones.

10. Add the new FDs to the design.

11. End.

For example :
Suppose we have a connectivity matrix :

	A1	A2	A3	A4
A1	0	0	0	0
A2	0	0	0	0
A3	0	0	0	1
A4	1	1	0	0

where A1 = CHAIRMAN,
A2 = TOTAL-STUDENT,
A3 = STUDENT#,
A4 = DEPARTMENT#.

Applying step 3 - 7 of the above transitive closure procedure, we derive a new connectivity matrix :

	A1	A2	A3	A4
A1	0	0	0	0
A2	0	0	0	0
A3	1	1	0	1
A4	1	1	0	0

Comparing with the original matrix, the following new FDs are derived :

STUDENT# <<--> CHAIRMAN
STUDENT# <<--> TOTAL-STUDENT

Both of the new FDs derived in this example are semantically meaningless. The next step then is to eliminate those semantically meaningless FDs by replacing "1" with "0" in the corresponding positions in the matrix. The modified matrix is shown below :

	A1	A2	A3	A4
A1	0	0	0	0
A2	0	0	0	0
A3	0	0	0	1
A4	1	1	0	0

This connectivity matrix allows us to derive the following set of ERs :

(STUDENT#, DEPARTMENT#)
(DEPARTMENT#, CHAIRMAN)
(DEPARTMENT#, TOTAL-STUDENT)

The procedure presented above allows us to automize the determination of transitive closures. However, some semantically meaningless FDs may also be produced by the procedure. It is necessary to check the semantic meaning of all the FDs derived, so that there will not be any redundant FDs in the design.

4.2 REDUCTION OF THE NUMBER OF ELEMENTARY RELATIONS

In practical cases, the designer may have thousands of ERs to describe the real world portion. Therefore, it is necessary to reduce the number of Elementary Relations (ERs).

The following procedures are used to reduce the number of ERs as much as possible [12].

1. Select all ERs having a non-key attribute occurring solely in the selected ER.
2. Create subsets of the ERs from step 1 such that all ERs within a subset have identical keys.
3. Each subset obtained in step 2 is replaced and represented by a single ER form.

4. End.

The following example demonstrates how this algorithm works :

Suppose we have the following set of ERs to describe the real world :

```
ER1( STUDENT#, LAST-NAME)
ER2( STUDENT#, FIRST-NAME)
ER3( STUDENT#, STATUS)
ER4( STUDENT#, ADDRESS)
ER5( STUDENT#, DEPARTMENT#)
ER6( STUDENT#, PROFESSOR#)
ER7( DEPARTMENT#, PROFESSOR#)
ER8( DEPARTMENT#, CHAIRMAN)
ER9( DEPARTMENT#, TOTAL-STUDENT)
```

where STUDENT# AND DEPARTMENT# are key attributes.

Step 1. The ERs having a non-key attribute occurring solely in a single ER are :

ER1, ER2, ER3, ER4, ER8, ER9

Step 2. Obtain a first subset with ERs whose key is STUDENT#, and a second subset with ERs whose key is DEPARTMENT# :

```
STUDENT# <<--> LAST-NAME, FIRST-NAME, STATUS,
               ADDRESS
DEPARTMENT# <<--> CHAIRMAN, TOTAL-STUDENT
```

Step 3. Replace the subsets obtained in step 2 by the following ERs :

```
ER10( STUDENT#, X1)
ER11( DEPARTMENT#, X2)
```

where

X1 = LAST-NAME, FIRST-NAME, STATUS, ADDRESS
X2 = CHAIRMAN, TOTAL-STUDENT.

The real world can be defined by fewer ERs :

```
ER5( STUDENT#, DEPARTMENT#)
ER6( STUDENT#, PROFESSOR#)
ER7( DEPARTMENT#, PROFESSOR#)
ER10( STUDENT#, X1)
ER11( DEPARTMENT#, X2)
```

4.3 SEARCH FOR REDUNDANCY

Data elements appearing as attributes in more than one record are potentially redundant attributes. The next procedure is used to search for those identical data elements, display the list, and delete the undesired ones.

1. Repeat, until every attribute in all records has been checked.
2. Check if there exists any attribute in other records which is identical to the one being checked.
3. If any redundant attribute exists then :
    a. Display the redundant attribute.
    b. Decide whether it should be deleted.
    c. Delete the redundant attributes according to the instructions given by the designer.
4. End.

Data elements are stored by the computer program in the form of arrays (which are referred as records in the algorithm). The purpose of the above algorithm is to

pick up one data element at a time, search through the data elements in other arrays to check if there is any identical data element to the one being checked. If identical data element exists, the program will display the potential redundant element for both arrays, and ask the designer to decide whether the data element should be in both or only one of the arrays.

4.4 MAPPING BETWEEN KEYS

Having determined the contents for each record, it is necessary to generate the relationship among the records by the mapping of keys.

The following procedures will determine the mapping type existing among the records.

1. Repeat, until done with all records.
2. Pick up the key attribute for all records.
3. Determine the mapping type.
    a. Display the possible mappings.
    b. Decide if mapping exists. If not, go to step 4.
    c. Select the type of mapping. (i.e. M:M, M:1, 1:1)
    d. Add the mapping to the design.
4. End.

The second data element in each array (record) is the key attribute (The first data element is the name of the record). The computer program picks up the second element in each array to form a list of key attributes. It will then display two key attributes each time, ask the designer to decide whether any mapping exists between the two key attributes being displayed. If mapping does exist, a list of possible mapping types is displayed for selection. After the designer has selected the mapping type, the relation for the mapping will be generated in form of a new relation set, and stored as a new array in the design.

4.5 GENERATE RELATIONS FOR TRIVIAL DEPENDENCE

The FD of the form

A --> B

where B is a subset of A is called a trivial dependence. For all entity sets, it is necessary to check for the existence of the dependence of this type, so it won't remain neglected.

Two occasions for the occurrence of trivial dependence are

1. A simple key attribute is a subset of a certain set of composite key attributes.

2. A set of composite key attributes is a subset of some other set of composite key attributes.

For example :

Case 1 :

(STUDENT#) is trivially dependent on
(STUDENT#, COURSE#)

Case 2 :

(STUDENT#, COURSE#) is trivially dependent on
(STUDENT#, SEMESTER, COURSE#)

The following example shows the importance of the generation of relations for this kind of dependence :

Suppose we have the following FDs :

(SEMESTER, COURSE#) <<--> COURSE-TITLE
(SEMESTER, COURSE#, STUDENT#) <<--> GRADE

The trivial dependence can be derived :

(SEMESTER, COURSE#, STUDENT#) <<-->
(SEMESTER, COURSE#)

If we fail to generate relation for the above FD, when we determine the transitive closure, the FD :

(SEMESTER, COURSE#, STUDENT#) <<--> COURSE-TITLE

will not be derived, because it represents the composition of

(SEMESTER, COURSE#, STUDENT#) <<-->
(SEMESTER, COURSE#) <<-->
COURSE-TITLE

The following procedures will generate relations for those trivial dependencies :

1. Repeat, until done with all records.
2. Pick up the key attribute for all the records.
3. If the key attribute selected is a subset of any other composite keys, then generate relation for the FD.
4. End.

The computer program picks up all the key attributes for all the arrays, and checks, one at a time, to see if the key attribute or the set of composite key attributes being checked is a subset of any other set of composite key attributes. If trivial dependencies exist, the relations for the dependencies will be generated in form of new relation sets, and stored as new arrays in the design.

5. CASE STUDY - DATA BASE DESIGN IN A BANKING ENVIRONMENT

The case study is adopted from "Structured Techniques for Design, Performance, and Management" by S. Atre. A brief description of the environment, the demonstration of design, and the discussion are included.

5.1 THE ENVIRONMENT

" Popular Bank " has a number of branch offices scattered throughout the city and the suburbs. A customer may walk into any branch and open an account. The customer is assigned a customer identification number (CID number) with the first account at the bank. The customer may open a number of accounts at the bank, but his/her CID number is not changed. The bank maintains checking accounts, saving accounts, loan accounts (customers must return loan with interest), and mortgage accounts (customers must make regular payments with interest).

The descriptions of the reports and transactions are as follows :

1. Branch Manager Report (Exception) - Every branch has a branch manager. At a certain point in time a branch has only one branch manager. A Branch Manager Report (Exception) is printed for the branch manager on a daily basis. The report consists of exceptional transactions only.

2. Branch Manager Report (Weekly Exception

Summary) - Another type of report is the Weekly Exception Summary. This report is not based on the individual accounts but is a summary. The two reports mentioned so far may be considered as batch reports.

3. Branch Loan Status - The branch manager also receives a Branch Loan Status . This report helps the branch manager to keep track of the loans given at the branch. It also serves as a source of new ideas for making loans at the branch attractive.

4. Teller Cash Drawer - The bank employs a number of tellers who cash checks and make deposits and withdrawals for the customers. The tellers are rotated among the branches, but it is assumed that once a teller is sent to a branch he/she will stay at that branch the whole day. A cash drawer is recorded every day for every teller. It keeps track of all the money flow for that day, for that teller, and at that branch. Teller Cash Drawer may be considered as a batch application.

5. Teller Audit Report - The tellers are audited periodically, as well as for specific reasons. Every teller has a unique identification number called a teller number. Based on experience and on previous audit records, a teller may withdraw only up to a certain amount of money for a customer. If the customer wants to withdraw more money than the specific teller's maximum allowable upper limit, the teller has to request another teller who can withdraw the amount to do so. The Teller Audit Report, which could be considered as a batch report, is sent to the central audit location for the bank.

6. Inquiry Transaction - It may happen that a customer walks into a branch and wants to deposit or withdraw a certain amount of money. Or he/she may want to know the balance of a specific account. The customer may not be able to provide the customer identification number (CID) or the account number(s). In such a situation the teller should be able to provide the information needed by the customer. This

online transaction is only an INQUIRY transaction ; it does not deposit or withdraw any money from any of the accounts but only provides information about the status of the accounts.

7. Deposit/Withdrawal Transaction - These are two types of transactions. Type1 is DEPOSIT, and Type2 is WITHDRAWAL. These two are on-line transactions.

After studying the environment, the following assumptions may be made :

Assumptions about the environment of " Popular Bank " :

1. The customer identification number (CID number) is unique.

2. Account numbers are allotted bankwide, that is, the account numbers are unique. The account number uniquely identifies the account type , the branch in which the particular account was opened, and the customer name, that is, the same account number will not be given to two or more accounts. The account number also uniquely identifies the customer.

3. Teller numbers are unique.

4. A teller can be assigned to different branches on different days, but once assigned to a branch he/she works there the whole day.

5. The account number, with transaction date and transaction time, uniquely identifies the transaction.

6. The action to be taken and the remarks to be registered for an exception are dependent on the reason code and on the account number, that is, on the customer for whom an exceptional action has to be taken.

7. Branch number, together with reason code, the date of start of report, and the date of end of report, uniquely identifies the total number of transactions for the weekly exception summary.

8. The loan number uniquely identifies the specific loan. The loan number is different from the account number. Account and loan are two separate entities.

9. A customer can have many account numbers at the same branch.

10. " Transaction Type " and " Transaction Code " are synonymous.

11. An INQUIRY transaction is only an information retrieval transaction.

12. If the teller worked in several branches during the audit period, a report for each branch will be printed. For each audit period for a given teller, there is only one reason code for the audit. Only the count of transactions and the largest dollar amount handled by teller being audited are taken into consideration.

13. The Exception Report and Weekly Exception Summary are for the branch where the account is kept. The reason is that the branch manager where the transaction was made is not too concerned if the account in another branch is overdrawn or if saving exceeds a certain limit. The branch manager is interested, however, if these things happen to his/her bank's accounts, even if the transaction triggering them takes place elsewhere.

14. Two types of transactions are recorded. Type1 is the DEPOSIT transaction , and Type2 is the WITHDRAWAL transaction. An INQUIRY transaction is not recorded.

15. At a given point in time a branch has only one branch manager.

A list of all data elements referenced in the reports in alphabetical order is :

ACCT#	Account number
ACCT-TYPE	Account type
ACTION-CODE	Action to be taken in exceptions
AMNT	Amount involved in particular transaction
AMNT-TTL	Amount involved in Weekly Exception Summary report
ADT-RESN	Reason code for audit
BALANCE	Balance of an account
BRANCH-NAME	Name of branch
BRANCH#	Number of branch
CASH-DSPNSD	Total cash dispensed during day
CASH-RCVD	Total cash received during day
CID	Customer identification number
COH-EOD	Cash on hand at the end of the day
COH-SOD	Cash on hand at the start of the day
CUST-ADDR	Customer address
CUST-DOB	Customer date of birth

CUST-NAME	Customer name
DATE	A business day
LAGST-AMNT-TYP1	Largest amount of transaction type1
LAGST-AMNT-TYP2	Largest amount of transaction type2
LOAN-ASSGND	Amount of money assigned as a loan
LOAN#	Loan number
LOAN-TYP	Loan type
MGR-NAME	Manager name
RESN-CODE	Reason code for the exception
TELLER-NAME	Teller name
TELLER#	Teller number
X-NO	Transaction number

5.2 THE DESIGN

The database design process is an iterative process. This software gives the user the capability to enter data and to control the editing process and logical design. It is basically a menu-driven process. A hierarchy of menus is provided, as shown in the following figures :

1. CONCEPTUAL DESIGN

2. STRUCTURE REFINEMENT

3. DESIGN RESULT

Figure 6.1 Initial menu

1. DATA ENTRY
2. REEDITING
3. SEARCH REDUNDANCY
4. MAPPING BETWEEN KEYS
5. DISPLAY DESIGN DIAGNOSTIC
6. EXIT

Figure 6.1.1 Menu for Conceptual Design

1. RECORD WITH SIMPLE KEY
2. RECORD WITH COMPOSITE KEYS

Figure 6.1.1a Menu for deciding record type

1. KEY WORD IN LIST
2. RELATIONSHIP SETS
3. ELEMENTARY RELATIONS

Figure 6.1.1b Menu for display

1. TRANSITIVITY REMOVAL

2. REDUCING THE NO. OF ERs

3. DISPLAY DESIGN DIAGNOSTIC

4. EXIT

Figure 6.1.2 Menu for Structure Refinement

The following is the demonstration of the design process in the " Popular Bank " case study, with explanations corresponding to the structured procedures discussed previously.

Step 1.1 Combination of Local Views

The designer enters local view 1 through n in step 1.1. Modifications to local views are made in step 1.2 to accommodate the various needs of users.

To enter the local views, the designer first selects the Conceptual Design function in the menu shown in fig 6.1 . The computer will respond with a menu as in fig 6.1.1. The designer then selects the first function to enter the records with simple key domain. Having done with the data entries, the designer simply types "/" to exit. After data are entered, the computer will display each record and ask the designer to enter the key domain.

Again the computer will display each record with key domain in the second column of that record (The record name is in the first column). The process is shown below :

```
SELECT THE TYPE OF RECORD
1. RECORD WITH SIMPLE KEY

2. RECORD WITH COMPOSITE KEYS
 1
 Enter Record 1
branch
 Enter Record 2
teller

  ALL THE RECORDS ENTERED ARE AS FOLLOWS :
+-+-------+-------+-------+
| |BRANCH |TELLER |CUSTOM |
+-+-------+-------+-------+
  For Record : 1
  branch#
  branchname
  mgrname
  /
  For Record : 2
  teller#
  tellername
  /
```

```
 ALL THE ATTRIBUTES ENTERED ARE AS FOLLOWS :
 For Record : 1
+-------+---------+----------+---------+
|BRANCH |BRANCH # |BRANCHNAME |MGRNAME |
+-------+---------+----------+---------+
 For Record : 2
+-------+---------+-----------+
|TELLER |TELLER # |TELLERNAME |
+-------+---------+-----------+


 ENTER THE KEY FOR EACH RECORD .
 For Record : 1
+-------+---------+----------+---------+
|BRANCH |BRANCH # |BRANCHNAME |MGRNAME |
+-------+---------+----------+---------+
 Enter Key Domain : branch#
 THE RECORD WITH KEY DOMAIN IN THE SECOND COLUMN IS DISPLAYED
+-------+---------+----------+---------+
|BRANCH |BRANCH # |BRANCHNAME |MGRNAME |
+-------+---------+----------+---------+
 For Record : 2
+-------+---------+-----------+
|TELLER |TELLER # |TELLERNAME |
+-------+---------+-----------+
 Enter Key Domain : teller#
 THE RECORD WITH KEY DOMAIN IN THE SECOND COLUMN IS DISPLAYED
+-------+---------+-----------+
|TELLER |TELLER # |TELLERNAME |
+-------+---------+-----------+
```

For records with composite keys, the process is different in determining

the key domains. The computer will ask the designer to enter the number of key domains (n), then ask the designer to enter data with key domains the first n data entries. As shown below :

```
 For Record : 6
First Enter The Number of Keys In This Record: 3
Please Enter The Key Domains In The First 3
Entries Of The Data Inputs
 teller#
 branch#
 date
 coh-eod
 coh-sod
 /
```

```
 For Record : 6
+-------+---------+---------+-----+---------+---------+
|DRAWER |TELLER # |BRANCH # |DATE |COH - EOD |COH - SOD |
+-------+---------+---------+-----+---------+---------+
```

Step 1.2 Reediting

This step serves the purpose of making changes to the data entered previously. The following examples demonstrate the editing capabilities including three editing features :

1. Inserting a new attribute
2. Modifying an old attribute
3. Switching keys

```
 Select The Number Of The Following Editing
   1. Insert New Attribute
   2. Delete Attribute
   3. Modification Of Attribute Names
   4. Switch Key Attribute
 1
+-------+----------+---------+
|BRANCH |BRANCHNAME |BRANCH # |
+-------+----------+---------+
Enter The Name Of The Attribue To Be Inserted
 mgrname
+-------+----------+---------+---------+
|BRANCH |BRANCHNAME |BRANCH # |MGRNAME  |
+-------+----------+---------+---------+
```

```
 Modification Completed ( For This Record ) ?
 Enter Y / N
  n
  Select The Number Of The Following Editing
    1. Insert New Attribute
    2. Delete Attribute
    3. Modification Of Attribute Names
    4. Switch Key Attribute
  3
+-------+----------+---------+---------+
|BRANCH |BRANCHNAME |BRANCH # |MGRNAME |
+-------+----------+---------+---------+
Enter The Name Of The Attribute To Be Modified
 mgrname
Change From ( MGRNAME ) To :
 mgr-name
+-------+----------+---------+-----------+
|BRANCH |BRANCHNAME |BRANCH # |MGR - NAME |
+-------+----------+---------+-----------+
Modification Completed ( For This Record ) ?
Enter Y / N
 n
 Select The Number Of The Following Editing
   1. Insert New Attribute
   2. Delete Attribute
   3. Modification Of Attribute Names
   4. Switch Key Attribute
 4
+-------+----------+---------+-----------+
|BRANCH |BRANCHNAME |BRANCH # |MGR - NAME |
+-------+----------+---------+-----------+
Enter The New Key Attribute Name
 branch#
+-------+---------+----------+-----------+
|BRANCH |BRANCH # |BRANCHNAME |MGR - NAME |
+-------+---------+----------+-----------+
```

Step 1.3 Search for Redundancy

To remove the potential redundant attributes in a record, simply select the Search for Redundancy function. The following demonstrates this function :

```
Attribute BRANCHNAME  In Record BRANCH
 Is The Same As Record LOAN
+-------+---------+----------+----------+-+-+-+-+-+-+-+-+
|BRANCH |BRANCH # |BRANCHNAME |MGR - NAME | | | | | | | | |
+-------+---------+----------+----------+-+-+-+-+-+-+-+-+
+-----+------+--------+----------+-+-+-+-+-+-+-+-+-+-+-+
|LOAN |LOAN # |LOANTYPE |BRANCHNAME | | | | | | | | | |
+-----+------+--------+----------+-+-+-+-+-+-+-+-+-+-+-+
Should This Attribute Be In Both Records? Enter Y / N n
 Which Record Shall This Attribute In : branch
Attribute Deleted In Record LOAN
+-----+------+--------+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|LOAN |LOAN # |LOANTYPE | | | | | | | | | | | | | | |
+-----+------+--------+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Step 1.4 Mapping between Keys

To determine the mapping type between keys, select the Mapping between Keys function in the menu. An example of this function is as follows :

Computer responds

```
Does Mapping Exist Between Key
BRANCH #  And CID  ? (Y/N)
```

The designer responds

Y

Computer responds

```
Determine The Mapping Type.(ie. 1,2,3,4)
1. 1 : 1 ( < -- > ) mapping
2. M : 1 ( <<-- > ) mapping
3. 1 : M ( < -->> ) mapping
4. M : M ( <<-->> ) mapping
```

The designer responds

3

After the mapping type has been determined, the relationships will be generated by creating new relationship sets for the relationships, and adding them to the design.

Step 1.5 Display of Design Diagnostic

This function can be used to review the design. The following examples demonstrate the three display capabilities :

KEY WORD IN LIST

- KEY -	- ATTRIBUTE -
BRANCH #	
	BRANCHNAME
	MGRNAME
TELLER #	
	TELLERNAME
CID	
	CUSTOMERNAME
	CUSTOMERDOB
	CUST - ADDR
LOAN #	
	LOANTYPE
	LOANASSGND
ACCT #	
	ACCTTYPE
	BALANCE

RELATION SETS

```
SELECT THE FOLLOWING DISPLAY
1. KEY WORD IN LIST
2. RELATIONSHIP SETS
3. ELEMENTARY RELATIONS
 2
BRANCH (  BRANCH # ,BRANCHNAME ,MGRNAME  )
TELLER (  TELLER # ,TELLERNAME  )
CUSTOM (  CID ,CUSTOMERNAME ,CUSTOMERDOB ,CUST - ADDR  )
LOAN (  LOAN # ,LOANTYPE ,LOANASSGND  )
ACCOUNT (  ACCT # ,ACCTTYPE ,BALANCE  )
1(      CID  , BRANCH #  )
2(      LOAN #  , BRANCH #  )
3(      ACCT #  , BRANCH #  )
4(      LOAN #  , CID  )
5(      ACCT #  , CID  )
DRAWER (  TELLER # ,BRANCH # ,DATE ,COH - EOD ,COH - SOD  )
TRANX (  ACCT # ,DATE ,TIME ,AMNT  )
```

Note that the relation sets with numeric number as their names are those relationships generated by the computer program during one of the following processes :

1. Mapping between keys.
2. Generating relations for trivial dependence.

ELEMENTARY RELATIONS

```
SELECT THE FOLLOWING DISPLAY
1. KEY WORD IN LIST
2. RELATIONSHIP SETS
3. ELEMENTARY RELATIONS
 3
ER1( BRANCH # , BRANCHNAME  )
ER2( BRANCH # , MGRNAME  )
ER3( TELLER # , TELLERNAME  )
ER4( CID , CUSTOMERNAME  )
ER5( CID , CUSTOMERDOB  )
ER6( CID , CUST - ADDR  )
ER7( LOAN # , LOANTYPE  )
ER8( LOAN # , LOANASSGND  )
ER9( ACCT # , ACCTTYPE  )
ER10( ACCT # , BALANCE  )
ER11( CID , BRANCH #  )
ER12( LOAN # , BRANCH #  )
ER13( ACCT # , BRANCH #  )
ER14( LOAN # , CID  )
ER15( ACCT # , CID  )
ER16( TELLER #    BRANCH #    DATE   , COH - EOD  )
ER17( TELLER #    BRANCH #    DATE   , COH - SOD  )
ER18( ACCT #    DATE    TIME   , AMNT  )
ER19( TELLER #    BRANCH #    DATE   , BRANCH #  )
ER20( TELLER #    BRANCH #    DATE   , TELLER #  )
ER21( ACCT #    DATE    TIME   , ACCT #  )
```

After the above steps are completed and satisfactorily edited, the designer proceeds to the following phase - Structure Refinement.

Step 2.1 Transitivity Removal

The following demonstration shows the process of

determining the transitive closure, selecting the semantically meaningful FDs, and removing the undesirable ones :

```
Is The Following Dependency Semantically Meaningful ? (Y / N)
TELLER #    BRANCH #    DATE    -- > BRANCHNAME
 n
Is The Following Dependency Semantically Meaningful ? (Y / N)
TELLER #    BRANCH #    DATE    -- > MGRNAME
 n
Is The Following Dependency Semantically Meaningful ? (Y / N)
TELLER #    BRANCH #    DATE    -- > TELLERNAME
 n
Is The Following Dependency Semantically Meaningful ? (Y / N)
ACCT #    DATE    TIME    -- > BRANCH #
 y
Is The Following Dependency Semantically Meaningful ? (Y / N)
ACCT #    DATE    TIME    -- > BRANCHNAME
 n
Is The Following Dependency Semantically Meaningful ? (Y / N)
ACCT #    DATE    TIME    -- > MGRNAME
 n
Is The Following Dependency Semantically Meaningful ? (Y / N)
ACCT #    DATE    TIME    -- > CID
 y
Is The Following Dependency Semantically Meaningful ? (Y / N)
ACCT #    DATE    TIME    -- > CUSTOMERNAME
 n
Is The Following Dependency Semantically Meaningful ? (Y / N)
ACCT #    DATE    TIME    -- > CUSTOMERDOB
 n
```

After the above interactive process, the computer program will modify the connectivity matrix according to the instructions given by the designer. For those relationships determined to be semantically meaningless

by the designer, the computer program will replace the "1" with "0" in the corresponding positions of the connectivity matrix to indicate that the relationships have been removed.

Step 2.2 Reducing the Number of ERs

This step is the final step in structure refinement. A matrix indicating the relationships among data elements is displayed :

	BRANCH #	BRANCH - ADDR	MGR - NAME	BRANCH - NAME
BRANCH #	0	1	1	1
BRANCH - ADDR	0	0	0	0
MGR - NAME	0	0	0	0
BRANCH - NAME	0	0	0	0

Having done the Conceptual Design and Structure Refinement, design results can be discussed.

step 3.0 Design Results

The design results for this case study are displayed below :

```
(   BRANCH #   , BRANCHNAME   , MGRNAME   ,   )
(   CID   , CUSTOMERNAME   , CUSTOMERDOB   , CUST - ADDR   ,   )
(   LOAN #   , LOANTYPE   , LOANASSGND   ,   )
(   ACCT #   , ACCTTYPE   , BALANCE   ,   )
(   TELLER #   BRANCH #   DATE   , COH - EOD   , COH - SOD   , )
( TELLER # , TELLERNAME )
CID  <<--> BRANCH #
LOAN #  <<--> BRANCH #
LOAN #  <<--> CID
ACCT #  <<--> BRANCH #
ACCT #  <<--> CID
TELLER #    BRANCH #    DATE    <<--> BRANCH #
TELLER #    BRANCH #    DATE    <<--> TELLER #
ACCT #    DATE    TIME    <<--> BRANCH #
ACCT #    DATE    TIME    <<--> CID
ACCT #    DATE    TIME    <<--> ACCT #
( ACCT #    DATE    TIME   ,AMNT  )
DESIGN COMPLETED
```

It should be noticed that the data elements encompassed by parentheses are the relations. The data elements with " <<--> " between them represents the mappings between records.

We can easily draw the following logical model by rearranging the design results :

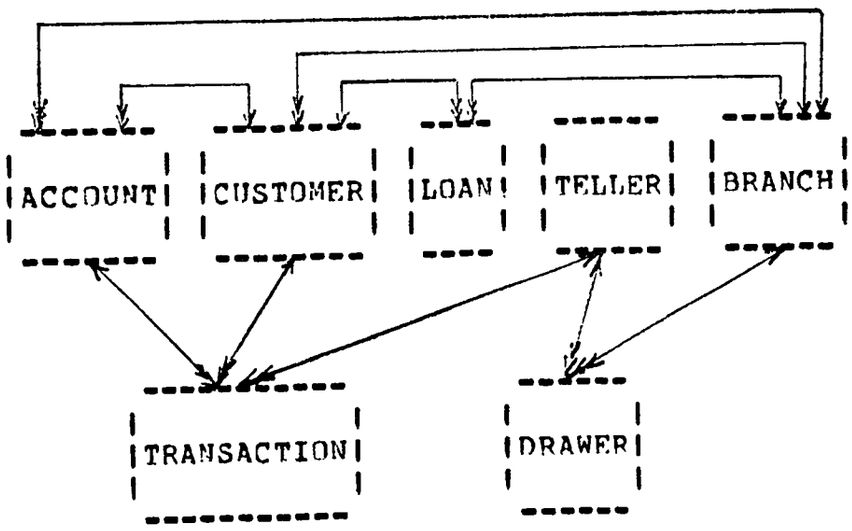


Figure 6.2 The Logical Model For
Popular Bank Data Base Design

And we have following relations :

BRANCH (BRANCH#, BRANCH-NAME, MGR-NAME)
CUSTOMER (CID, CUST-NAME, CUST-DOB, CUST-ADDR)
LOAN (LOAN#, LOAN-TYPE, LOAN-ASSGND)
ACCOUNT (ACCT#, ACCT-TYPE, BALANCE)
TELLER (TELLER#, TELLER-NAME)
DRAWER (TELLER#, BRANCH#, DATE, COH-EOD, COH-SOD)
TRANSACTION (ACCT#, DATE, TIME, AMNT)

Comparing with the design result in reference [1], the conceptual model developed by the computer program has two extra relations :

1. ACCOUNT <<-> BRANCH - This relation is redundant because BRANCH is transitively dependent on ACCOUNT. It can be derived by the following two relations :

ACCOUNT <<-> CUSTOMER
CUSTOMER <<--> BRANCH

However, the relation ACCOUNT <<--> BRANCH is semantically meaningful in establishing connection between ACCOUNT and BRANCH. Therefore, the branch where the account is kept can be obtained directly from on-line inquiry, which greatly speeds up inquiry transaction.

2. LOAN <<--> BRANCH - The similar situation applies to this relation. It is redundant because it can be derived from the following relations :

LOAN <<--> CUSTOMER
CUSTOMER <<-> BRANCH

Again, this relation establishes the connection between LOAN and BRANCH, and speeds up inquiry transaction.

6. CONCLUSION

Data base design has been described as an intuitive and artistic process, typically, iterative. During each iteration, the goal is to approach more closely an acceptable design. Thus a design will be developed and reviewed, the defects will be identified, and the design steps will be repeated until no major defects can be found by either users and designers.

The methodology presented in this thesis has shown that it is feasible to replace the intuitive approach to data base design with a systematic, computer-assisted approach. This allows the designer and analyst to concentrate more on the semantic problems for which no mechanized solution will be available. Alternative designs may also be evaluated much more easily. The computer-assistance also provides useful diagnostic reports telling the designer about which data items have been used so far in the design, and where redundancies are. The expected benefits are summarized below :

1. Presents information(such as design diagnostic) helpful to the designer which greatly reduces the tediousness of the design procedure.

2. Performs a more thorough and consistent analysis of data requirements and inevitable design trade-offs.

3. Improves design quality by :

a. Removing undesirable FDs.

b. Identifying and removing redundancies.

4. Shortens the design cycle by :

a. Reducing the number of design iterations.

b. Reducing the time for each design iteration.

c. Documenting the results, so the designer and the end users can obtain a more objective basis for dealing with differing perspectives.

It should be emphasized that, in the field of data base design, there are principles and tools, but they must always be used in conjunction with intuition and guided by experience.

REFERENCE

1. Atre, S., Data Base, Structured Techniques for Design, Performance, and Management. John Wiley, 1980.

2. Codd, E. F., "Extending the Relational Model to Capture More Meaning." In transaction on Database Systems, Vol. 4, No. 4, December 1979.

3. Date, C. J., An Introduction to Data Base Systems, Third Edition. Addison-wesley, 1981.

4. Engles, R. W., "A Tutorial on Database Organization." IBM Corp. Technical Report TR.00.2004, IBM, Poughkeepsie, N.Y., 1970.

5. Hubbard, G. U., Computer Assisted Data Base Design. Van Nostrand Reinhold, 1981.

6. IBM Corp., Data Base Design Aid (Version 2): Designer's Guide, Publication No. SH20-1627(1977).

7. IBM Corp., DBPROTOTYPE/II Program Description/Operation Manual, Publication No. SH20-1953(1978).

8. Jenkins, M. A., The Q'NIAL Reference Manual. Queen's University, Kinston, Canada, 1983.

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

10. Mealy, A. H., "Another Look at Data." in Proc. AFIPS 1967 Fall Joint Computer Conference, Vol. 31, AFIPS Press, Montvale, N.J., 1968

11. Schmidt, F. L. & Jenkins M. A. Data Systems - The NIAL approach. Queen's University, Kinston, Canada, 1982.

12. Tsichritzis, Dionysios C., & Lochovsky, Frederick H. , Data Models. Prentice-Hall International, 1982.

13. Ullman, Jeffrey D., Principles of Database

Systems. Computer Science Press, 1980.

14. Vetter, M. & Madison, R. N., Database Design Methodology. Prentice-Hall International, 1981.

15. Weiderhold, Gio, Database Design, Mcgraw-Hill, 1983.

APPENDIX A: THE DOCUMENTATION

The program is defined in the script file named DB.NDF, the contents of which are listed in the next section after the brief descriptions of the operations given below. (NDF is an acronym and a suffix used in the host system to mean Nial Definition File.).

1. **ENTRY** : This operation is used for the entering of the entity set names in the initial data entry.

2. **ATTR** : This operation is used for the entering of the attributes in the initial data entry.

3. **KEY** : This operation is used to identify the key domains for the records entered previously.

4. **NUM** : This operation calculates the number of all the data elements entered, which includes the number of records, the number of attributes, and the number of key domains for each record.

5. **TABLE** : This operation displays the KEY-WORD-IN-LIST.

6. **DISPL** : This operation is used to display the entity sets and relationship sets for all the data elements entered.

7. **MODI** : This operation is used to modify the data elements in a specific record. It performs the tasks including insert attributes, delete attributes, modify attribute names, and switch key domains.

8. **REEDIT** : This operation is used to make changes for all the records entered previously. It performs the tasks including inserting new records, deleting old records, and modifying old records (MODI).

9. **SRCH** : This operation searches redundant attributes existing in the records entered previously.

10. **MAP** : This operation determines the mapping types among key domains.

11. **GEN** : This operation generates relations for trivial dependencies.

12. **REL** : This operation is used to generate the ERs.

13. **ERR** : This operation displays all the ERs.

14. **ALT** : This operation generates a list of all data elements with every data element appearing only once.

15. **MTRX** : This operation is used to derive the connectivity matrix.

16. **FRAME** : This operation generates a frame which gives the connectivity matrix coordinates of reference.

17. **COMP** : This operation determines the compositions for the connectivity matrix.

18. **CLOSURE** : This operation determines transitive closure.

19. **SOL** : This operation is used to select, from among the transitive dependencies derived, the semantically meaningful ones.

20. **TRANS** : This operation is used to transform the NIAL expression of Boolean variables (NIAL use "o" "1" to represent truth and false) into common expression of Boolean variable (i.e. "0" "1") for the manipulation of Boolean logic.

21. **REDUCES** : This operation is used to reduce ERs according to the algorithm presented in section 4.2.

22. **DISPLAY** : This operation is used to display the records entered, their contents and

relations. It includes operation TABLE, DISPL, and ERR.

23. **Result** : This operation displays the design results.

For the demonstration of how the program works, refer to the CASE STUDY.

APPENDIX B: THE PROGRAM

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                          %
%                                          %
%   COMPUTER ASSISTED DATA BASE DESIGN     %
%                                          %
%  * * * * * * * * * * * * * * * * * * * * %
%                                          %
%      BY : CHIEN-CHUNG J. CHUANG          %
%                                          %
%             SEPTEMBER 1984               %
%                                          %
%              LEHIGH UNIV.                %
%                                          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
m := 1;

mr := 1;

i := 1;

check := 0;

c := ' ';

dd := ' ';

ddd := ' ';

dtd := ' ';

entry is op e(
a := ' ';
d := ' /';
e := 1 20 reshape ' ';
while scan a ~= scan d do
 writescreen. ' Enter Record ' link sketch i;
 a := readscreen '';
 b := phrase a;
if scan a ~= scan d then  c := c link b endif ;
```

```
 i := i + 1;
endwhile;
for j with count ((tally c) minus 1) do
cd := descan scan (j pick c);
 e := cd (0 j) place e;
endfor;
i := i minus 1;
writescreen ' ALL THE RECORDS ENTERED ARE AS FOLLOWS : ';
writescreen sketch e)

attr is op x(
if m ~= 1 then
mc := (m minus 1) drop (count ((tally c) minus 1));
m := tally c;
else m := tally c; mc := count (m minus 1) endif;
for at with mc do
 aa := ' ';
 cc := at pick c;
 ee := 1 20 reshape ' ';
 writescreen. ' For Record : ' link sketch at ;
 if check = 1 then
 ca := read 'First Enter The Number of Keys In This
             Record: ';
 writescreen. 'Please Enter The Key Domains In The First
               ' link sketch ca;
 writescreen 'Entries Of The Data Inputs';
 caa := caa link ca; endif;
    while scan aa ~= scan d do
     aa := readscreen ' ';
     bb := phrase aa;
     cc := cc link bb;
    endwhile;
    for jj with tell ((tally cc) minus 1) do
     gg := descan scan (jj pick cc);
     ee := gg (0 jj) place ee;
    endfor;
 ee := solitary ee;
 dd := dd link ee;
endfor;
writescreen ' ALL THE ATTRIBUTES ENTERED ARE
              AS FOLLOWS : ';

mm := (tally dd) minus 1;
for att with count mm do
disp := att pick dd;
writescreen. ' For Record : ' link sketch att;
writescreen sketch disp;
endfor)
```

```
key is op x(
pk := ' ';

if check ~= 1 then
writescreen ' ENTER THE KEY FOR EACH RECORD . ';endif;
if mr ~= 1 then
mcr := (mr minus 1) drop (count ((tally dd) minus 1));
mr := tally dd;
else mr := tally dd; mcr := count (mr minus 1) endif;
for att with mcr do
 disp := att pick dd;
 writescreen, ' For Record : ' link sketch att;

 writescreen sketch disp;
case check from
1 : hhh := solitary disp;
   dtd := dtd link hhh end
else  aaa := readscreen ' Enter Key Domain : ';
 temp := (0 1) pick disp;
 kkk := 1;
 while scan pk ~= scan aaa do
  pk := (0 kkk) pick disp;
  sn := (0 0) pick disp;
  kkk := kkk + 1;
 endwhile;
pk := phrase pk;
pck := pck link pk;
sn := phrase sn;
sns := sns link sn;
 if kkk ~= 1 then
 disp := pk (0 1) place disp;
 disp := temp (0 (kkk minus 1)) place disp
 endif;
writescreen ' THE RECORD WITH KEY DOMAIN IN THE SECOND
              COLUMN IS DISPLAYED ';
writescreen sketch disp;
eee := solitary disp;
ddd := ddd link eee endcase;
endfor; check := 0;)

num is op x(
an := (tally ddd) minus 1;
bn := (tally dtd) minus 1;
sss := '7';
iii := '7';
h := ' ';
for tn with count an do
 ii := 1;
 xx := tn pick ddd;
```

```
 repeat
 xxx := (0 ii) pick xx;
 ii := ii + 1;
 until scan xxx = scan h
 endrepeat;
iii := iii link ii;
endfor;
for rn with count bn do
  ss := 1;
  pp := rn pick dtd;
  repeat
  ppp := (0 ss) pick pp;
  ss := ss + 1;
  until scan ppp = scan h
  endrepeat;
sss := sss link ss;

endfor;

)

table is op x(
writescreen '* * * * * * * * * * * * * * * * * * * * *';
writescreen '                                         *';
writescreen '*                                         ';
writescreen '                                         *';
writescreen '*    - KEY -           - ATTRIBUTE -      ';
writescreen '                                         *';
writescreen '*                                         ';
writescreen '   = = = = = = = = = = = = = = = = = =   *';
writescreen '*                                         ';
writescreen '                                         *';
for tn with count an do
 xx := tn pick ddd;
 for tnn with count ((tn pick iii) minus 2) do
  xxx := (0 tnn) pick xx;
  if tnn = 1 then
  writescreen. '*      ' link sketch xxx;
  writescreen'                                        *';
  else
  writescreen. '*                          ' link sketch
                xxx;
  writescreen'                                        *';
  endif;
 endfor;
endfor;
for rn with count bn do
  pp := rn pick dtd;
  rr := rn pick caa;
```

```
 for rnn with count((rn pick sss) minus 2) do
   ppp := (0 rnn) pick pp;
   if rnn <= rr then
   writescreen. '*      ' link sketch ppp;                    *';
writescreen'
   else                                          ' link
   writescreen. '*
                sketch ppp;                                    *';
writescreen '
   endif;
 endfor;
endfor;                                                        ';
writescreen '*                                                 *';
writescreen '
writescreen '* * * * * * * * * * * * * * * * * * * * * * * * *')

displ is op x(
iii := iii minus 1;
for gy with count an do
 ggy := '( ';
 xg := gy pick ddd;
 mm := gy pick iii;
 for xy with tell mm do
  gg := (0 xy) pick xg;
  if xy = 0 then xgy := gg ; gg := ' ' endif;
  ggy := ggy link gg;
  if (xy ~= (mm minus 1)) and (xy ~= 0) then
  ggy := ggy link ',' endif;
 endfor;
 writescreen. sketch xgy link sketch ggy link ' )';
endfor;
iii := iii + 1;
sss := sss minus 1;
for w with count bn do
 ggy := '( ';
 wy := w pick dtd;
 nn := w pick sss;
 ny := w pick caa;
 for ww with tell nn do
  kk := (0 ww) pick wy;
  if ww = 0 then xgy := kk ; kk := ' ' endif;
  ggy := ggy link kk;
  if (ww ~= (nn minus 1)) and (ww ~= 0) then
  ggy := ggy link ',' endif;
  endfor;
 writescreen. sketch xgy link sketch ggy link ' )';
 endfor;
 sss := sss + 1;)
```

```
modi is op x(
 writescreen 'Select The Type Of Record Set';
    writescreen '1. Record With Simple Key';
    writescreen '2. Record With Compound keys';
    a := read ' ';
    If a = 1 then
    repeat
      yy := 0;
      writescreen ' Enter The Record To Be Modified';
      rr := readscreen ' ';
      repeat
       yy := yy + 1;
       uu := yy pick ddd;
       ppp := (0 0) pick uu;
      until (scan ppp = scan rr) or (yy = an)  endrepeat;
       writescreen sketch uu;
      repeat
       writescreen ' Select The Number Of The
                     Following Editing';
  writescreen '   1. Insert New Attribute                ';
  writescreen '   2. Delete Attribute                    ';
  writescreen '   3. Modification Of Attribute Names     ';
  writescreen '   4. Switch Key Attribute                ';
       ed := read ' ';
       case ed from
       1 : writescreen sketch uu;
           writescreen 'Enter The Name Of The Attribue To
                        Be Inserted';
           ins := readscreen ' ';
           ins := descan scan ins;
           ff := yy pick iii;
           ff := ff minus 1;
           uu := ins (0 ff) place uu;
           ff := ff + 2 ;
           iii := ff yy place iii end
       2 : writescreen sketch uu;
           writescreen 'Enter The Name Of The Attribute
                        To Be Deleted';
           de := readscreen ' ';
           ff := 0;
           repeat
             ff := ff + 1;
             qq := (0 ff) pick uu;
           until scan de = scan qq endrepeat;
           repeat
             fff := ff + 1;
             ins := (0 fff) pick uu;
             uu := ins (0 ff) place uu;
             ff := ff + 1;
```

```
      until scan ins = scan ' ' endrepeat;
      gf := yy pick iii;
      gf := gf minus 1;
      iii := gf yy place iii;
      uu := ' ' (0 fff) place uu end
  3 : writescreen sketch uu;
      writescreen 'Enter The Name Of The Attribute
                   To Be Modified';
      mo := readscreen ' ';
      ff := 0;
      repeat
        ff := ff + 1;
        qr := (0 ff) pick uu;
      until scan mo = scan qr endrepeat;
      mo := descan scan mo;
      writescreen. 'Change From ( ' link sketch mo
                    link ') To:';
      po := readscreen ' ';
      po := descan scan po;
      uu := po (0 ff) place uu end
  4 : writescreen sketch uu;
      writescreen'Enter The New Key Attribute Name';
      mo := readscreen ' ';
      ff := 0;
      repeat
      ff := ff + 1;
      qr := (0 ff) pick uu;
      until scan mo = scan qr endrepeat;
      temp := (0 1) pick uu;
       uu := temp (0 ff) place uu;
       uu := qr (0 1) place uu;
       for af with count ((tally pck) minus 1) do
        ab := af pick pck;
        if scan ab = scan temp then
        qr := phrase qr;
        pck := qr af place pck;endif;
       endfor end
   else writescreen'No Such Modification Capability?
                    Try Again!'
   endcase;
  ddd := uu yy place ddd;
  writescreen sketch uu;
  writescreen 'Modification Completed
               (For This Record) ?';
  writescreen 'Enter Y / N ';
  ch := readscreen ' ';
until scan ch = scan 'y' endrepeat;
num 0;
displ 0;
```

```
      writescreen 'Modification Completed
                   (For Record Set)?';
      chh := readscreen ' ';
   until scan chh = scan 'y' endrepeat ;
Else repeat
 yy := 0;
 writescreen ' Enter The Record To be Modified';
 rr := readscreen ' ';
 repeat
   yy := yy + 1;
   uu := yy pick dtd;
   ppp := (0 0) pick uu;
 until (scan ppp = scan rr) or (yy = bn) endrepeat;
 writescreen sketch uu;
 repeat
   writescreen ' Select The Number Of The Following
                Editing';
   writescreen '   1. Insert New Attribute          ';
   writescreen '   2. Delete Attribute              ';
   writescreen '   3. Modification Of Attribute Names';
   ed := read ' ';
   case ed from
   1 : writescreen sketch uu;
       writescreen 'Enter The Name Of The Attribute
                    To Be Inserted';
       ins := readscreen ' ';
       ins := descan scan ins;
       ff := (yy pick sss) minus 1;
       uu := ins (0 ff) place uu;
       ff := ff + 2;
       sss := ff yy place sss end
   2 : writescreen sketch uu;
       writescreen 'Enter The Name Of The Attribute
                    To Be Deleted';
       de := readscreen ' ';
       ff := 0;
       repeat
        ff := ff + 1;
        qq := (0 ff) pick uu;
       until scan de = scan qq endrepeat;
       repeat
        fff := ff + 1;
        ins := (0 fff) pick uu;
        uu := ins (0 ff) place uu;
        ff := ff + 1;
       until scan ins = scan ' ' endrepeat;
       gf := (yy pick sss) minus 1;
       sss := gf yy place sss;
       uu := ' ' (0 fff) place uu end
```

```
    3 : writescreen sketch uu;
        writescreen 'Enter The Name Of The Attribute
                     To Be Modified';
        mo := readscreen ' ';
        ff := 0;
        repeat
         ff := ff + 1;
         qr := (0 ff) pick uu;
        until scan mo = scan qr endrepeat;
        mo := descan scan mo;
        writescreen. 'Change From ( ' link sketch mo
                     link ') To :';
        po := readscreen ' ';
        po := descan scan po;
        uu := po (0 ff) place uu end
    else writescreen 'No Such Modification Capability?
                     Try Again!';
    endcase;
    dtd := uu yy place dtd;
    writescreen sketch uu;
    writescreen 'Modification Completed
                 (For This Record)?';
    ch := readscreen ' ';
    until scan ch = scan 'y' endrepeat;
    num 0;
    displ 0;
    writescreen 'Modification Completed
                 (For Record Set)?';
    chh := readscreen ' ';
    until scan chh = scan 'y' endrepeat;
    endif;
    )

reedit is op x(
chhh := ' ';
repeat
displ 0;
writescreen'Select The Number Of The Following Editing';
writescreen '    1. Insert New Record Set          ';
writescreen '    2. Delete Undesired Record        ';
writescreen'    3. Modifications Of Record sets Entered';
edd := read ' ';
case edd from
1 : writescreen'Select The Type Of The New Record Set. ';
    writescreen '1. Record Set With Simple Key.';
    writescreen '2. Record Set With Compound Keys.';
    a := read ' ';
    if a = 2 then check := 1; endif;
    e := entry e;
```

```
    attr 0;
    key 0 end
2 : displ 0;
    writescreen 'Enter The Name Of The Record
                 To Be Deleted';
    a := readscreen ' ';
     ss := ((tally ddd) minus 1);
       for x with count ss do
         b := x pick ddd;
         ba := (0 0) pick b;
          if scan ba = scan a then
           pk := (0 1) pick b;
             if x < ss then
              y := x + 1;
              yy := x;
              repeat
              bc := y pick ddd;
              ddd := bc yy place ddd;
              y := y + 1;
              yy := yy + 1;
              until y >= (ss + 1) endrepeat;
             endif;ddd := front ddd;
             z := pk find pck;
             if z ~= ((tally pck) minus 1) then
              zz := z + 1;
              zzz := z;
              repeat
               bg := zz pick pck;
               pck := bg zzz place pck;
               zz := zz + 1;
               zzz := zzz + 1;
              until zzz >= (tally pck) endrepeat;
              endif; pck := front pck;
          endif;
       endfor;
    ss := ((tally dtd) minus 1);
    for x with count ss do
     b := x pick dtd;
     ba := (0 0) pick b;
     if scan ba = scan a then
      z := x;
      if x < ss then
      y := x + 1;
      yy := x;
      repeat
      bc := y pick dtd;
      dtd := bc yy place dtd;
      y := y + 1;
      yy := yy + 1;
```

```
      until y >= (ss + 1) endrepeat;
      endif;dtd := front dtd;
       if z ~= ((tally caa) minus 1) then
       zz := z + 1;
       zzz := z;
       repeat
        bg := zz pick caa;
        caa := bg zzz place caa;
        zz := zz + 1;
        zzz := zzz + 1;
       until zz >= (tally caa) endrepeat;
       endif;caa := front caa;
     endif;
    endfor end
3 : modi 0 end
else writescreen'No Such Editing Capability? Try Again !'
endcase;
num 0;
Table 0;
displ 0;
writescreen ' Editing Completed ? Enter Y / N ';
chhh := readscreen ' ';
until scan chhh = scan 'y' endrepeat;
writescreen 'Display Of Editing Result : ';
num 0;
table 0;
displ 0)

srch is op x(
flag := 1;
iii := iii minus 2;
ss := count ((tally ddd) minus 1);
for sr with ss do
  che := sr pick ddd;
  raa := (0 0) pick che;
  nc := sr pick iii;
  for z with count nc do
    el := (0 z) pick che;
    for srr with (sr drop ss) do
      mat := srr pick ddd;
      ral := (0 0) pick mat;
      nm := srr pick iii;
      for zz with count nm do
        ell := (0 zz) pick mat;
        if scan el = scan ell then
         writescreen '    ';
         writescreen '    ';
         writescreen '    ';
         writescreen '    ';
```

```
          writescreen '    ';
          writescreen. 'Attribute ' link sketch el link '
                        In Record' link sketch raa;
          writescreen. ' Is The Same As Record '
                        link sketch ral;
          writescreen sketch che;
          writescreen sketch mat;
          ad := readscreen 'Should This Attribute Be In
                            Both Records? (Y/N)';
          if scan ad = scan 'n' then
          ra := readscreen ' Which Record Shall This
                             Attribute Be In: ';
          ra := phrase ra;
          if scan ra = scan raa then rae := ral;
             uu := mat; yy := srr; gf := nm; ff := zz;
          else uu := che; yy := sr; gf := nc;ff :=z;
               rae := raa; endif;
            repeat
              fff := ff + 1;
              ins := (0 fff) pick uu;
              uu := ins (0 ff) place uu;
              ff := ff + 1;
            until scan ins = scan ' ' endrepeat;
            gf := gf minus 1;
            iii := gf yy place iii;
            uu := ' ' (0 fff) place uu;
            ddd := uu yy place ddd;
          writescreen. 'Attribute Deleted In Record '
                        link sketch rae;
          writescreen sketch uu;
            endif;
            flag := 0;
           endif;
        endfor;
      endfor;
    endfor;
endfor;
if flag = 0 then flg := 0; endif;
iii := iii + 2;
displ 0)

map is op x(
e := 1;
for af with count ((tally pck) minus 1) do
  k := af pick pck;
  ox := af fold rest pck;
  for bf with count ((tally ox) minus 1) do
    j := bf pick ox;
    writescreen 'Does Mapping Exist Between Key';
```

```
      writescreen. sketch k link ' And ' link sketch j
                   link '? (Y/N)';
      cf := readscreen ' ';
      if scan cf = scan 'y' then
      writescreen 'Determine The Mapping Type.
                   (i.e. 1,2,3,4)';
      writescreen '1. 1 : 1 ( < -- > ) mapping';
      writescreen '2. M : 1 ( <<-- > ) mapping';
      writescreen '3. 1 : M ( < -->> ) mapping';
      writescreen '4. M : M ( <<-->> ) mapping';
      df := read ' ';
      case df from
      1 : vv := (solitary e) link (solitary k) link
                (solitary j) link (solitary ' ');
          ddd := ddd link solitary (1 4 reshape vv);
          e := e + 1;
          vv := (solitary e) link (solitary j) link
                (solitary k) link (solitary ' ');
          ddd := ddd link solitary (1 4 reshape vv);
          e := e + 1 end
      2 : vv := (solitary e) link (solitary k) link
                (solitary j) link (solitary ' ');
          ddd := ddd link solitary (1 4 reshape vv);
          e := e + 1 end
      3 : vv := (solitary e) link (solitary j) link
                (solitary k) link (solitary ' ');
          ddd := ddd link solitary (1 4 reshape vv);
          e := e + 1 end
      4 : vv := (solitary e) link (solitary k) link
                (solitary j) link (solitary k) link
                (solitary ' ');
          dtd := dtd link solitary (1 5 reshape vv);
          caa := caa link 2;
          e := e + 1;
          vv := (solitary e) link (solitary k) link
                (solitary j) link (solitary j) link
                (solitary ' ');
          dtd := dtd link solitary (1 5 reshape vv);
          caa := caa link 2;
          e := e + 1 end
      else writescreen 'Error ?' endcase;
    endif;
    endfor;
endfor;
num 0;)

gen is op x(
lag := 1;
kck := ' ';
```

```
nsn := ' ';
ckk := ' ';
for rn with count bn do
  pp := rn pick dtd;
  rr := rn pick caa;
  ns := phrase ((0 0) pick pp);
  nsn := nsn link ns;
    for rnn with count rr do
      ck := (0 rnn) pick pp;
      ck := phrase ck;
      ckk := ckk link ck;
    endfor;
  ckk := solitary (rest ckk);
  if (0 pick ckk) notin kck then
  kck := kck link ckk;endif;
endfor;
for zk with count ((tally pck) minus 1) do
  abc := zk pick pck;
  for kz with count ((tally kck) minus 1) do
    bcd := kz pick kck;
    xk := tally bcd;
    if abc in bcd then
    vv := bcd link (solitary abc) link (solitary ' ');
    xn := xk + 3;
    for ii with count e do
     jj := (1 xn) reshape ((solitary ii) link vv);
     if jj in dtd then lag := 0;endif;
    endfor;
    if lag ~= 0 then
    vv := (solitary e) link vv;
    dtd := dtd link solitary (1 xn reshape vv);
    caa := caa link xk;
    e := e + 1;
    endif;
    endif;
  endfor;
endfor;
num 0;
)

rel is op x(
ml := ' ';
ger := ' ';
lm := ' ';
mlm := ' ';
gr := ' ';
jjj := iii minus 2;
for co with count an do
  ca := co pick jjj;
```

```
  cm := co pick ddd;
  pk := (0 1) pick cm;
  pk := phrase pk;
  ge := solitary (pk link pk);
  ger := ger link ge;
    for it with (1 drop (count ca)) do
      atr := (0 it) pick cm;
      atr := phrase atr;
      re := pk link atr;
      re := solitary re;
      ml := ml link re;
    endfor;
endfor;
bbb := sss minus 2;
for oc with count bn do
  ac := oc pick bbb;
  mc := oc pick dtd;
  am := oc pick caa;
    for ti with count am do
      atr := (0 ti) pick mc;
      atr := (phrase atr) link (phrase ' ');
      gr := gr link atr;
    endfor;
  gr := solitary (rest gr);
  ke := gr link gr;
  ke := solitary ke;
  mlm := mlm link ke;
    for tii with count ac do
      if tii > am then
      atr := (0 tii) pick mc;
      atr := phrase atr;
      kr := solitary (gr link atr);
      lm := lm link kr;
      endif;
    endfor;
endfor;
)

err is op ml lm(
nml := (tally ml) minus 1;
for i with count nml do
  er := i pick ml;
  for j with 0 1 do
      dp := 0 pick er;
      dq := 1 pick er;
  endfor;
writescreen. 'ER' link sketch i link '( ' link
             sketch dp link ',' link sketch
             dq link ' )';
```

```
endfor;
lmn := (tally lm) minus 1;
for j with count lmn do
  re := j pick lm;
    for ii with 0 1 do
      pd := 0 pick re;
      dq := 1 pick re;
    endfor;
  i := i + 1;
  writescreen. 'ER' link sketch i link '( ' link
               sketch pd link ', ' link sketch
               dq link ' )';
endfor;
)

alt is op x(
gcc := ' ';
for x with count an do
  gl := x pick ddd;
  gc := x pick jjj;
  for xx with count gc do
    glc := (0 xx) pick gl;
    glc := phrase glc;
    gcc := gcc link glc;
  endfor;
endfor;
for y with count ((tally lm) minus 1) do
  lg := y pick lm;
  for yy with 0 1 do
    cg := yy pick lg;
    if yy = 0 then cg := solitary cg ; endif;
    gcc := gcc link cg;
  endfor;
endfor;
mlist := rest gcc;
nmi := tally mlist;
jj := ' ';
for j with tell nmi do
  x := j pick mlist;
  if x notin jj then
    if tally x > 1 then x := solitary x ; endif;
    jj := jj link x;
  endif;
endfor;
mlist := rest jj;)

mtrx is op x(
am := tally mlist;
mx := am am reshape 0;
```

```
for xa with tell am do
  qx := xa pick mlist;
  if tally qx > 1 then qx := solitary qx; endif;
  for xb with tell am do
    px := xb pick mlist;
    if tally px > 1 then px := solitary px; endif;
    qp := (qx link px);
    if (qp in ml) or (qp in ger) or (qp in lm)
       or (qp in mlm) then
    mx := 1 (xa xb) place mx; endif;
  endfor;
endfor;)


frame is op m mlist(
mix flip ("   hitch Mlist hitch flip (Mlist
          hitch rows M)))

comp is op mm(
WW:=(rows MM) eachleft eachright and (cols MM);
XM:=shape MM Reshape (or. flip link WW);
C:=MM or XM;
)

closure is op m(
Md:= M match 1;
I:= shape M reshape. o link. first shape M reshape 1;
C:= Md and I;
repeat
 mm := c;
 comp mm;
 C:=C and I;
until c = mm endrepeat;
tc:=1 (1 findall. Md or MM) placeall (shape M reshape 0);
frame tc Mlist;
)

sol is op x(
mt := tc > mx;
dex := 1 findall mt;
for xa with tell am do
  xq := xa pick mlist;
    for xb with tell am do
      xp := xb pick mlist;
      if (xa xb) in dex then
      writescreen 'Is The Following Dependency
                   Semantically Meaningful?(Y/N)';
    writescreen. sketch xq link ' -- > ' link sketch xp;
      ap := readscreen ' ';
```

```
      if scan ap = scan 'n' then
      tc := 0 (xa xb) place tc;
      endif;
      endif;
    endfor;
endfor;
)

trans is op mm(
1 (1 findall mm) placeall (shape mm reshape 0)
)

reduces is op tc(
mo:= tc match 1;
r := trans (tc > trans (shape tc reshape or cols mo));
r := trans (r >=. shape tc reshape sum rows tc);
r := r*trans(transpose. shape tc reshape sum cols r> 1);
)

displays is op x(
uh := 0;
num 0;
writescreen 'SELECT THE FOLLOWING DISPLAY';
writescreen '1. KEY WORD IN LIST';
writescreen '';
writescreen '2. RELATIONSHIP SETS';
writescreen '';
writescreen '3. ELEMENTARY RELATIONS';
ba := read ' ';
case ba from
1 : table 0 end
2 : displ 0 end
3 : gen 0; rel 0; err ml lm; uh := 1 end
else writescreen'ERROR ? TRY AGAIN !' endcase ;
)

search is op x(
repeat
flg := 1;
srch 0;
until flg ~= 0 endrepeat;
)

result is op x(
ec := 0;
ds := 1 findall r;
for id with tell ((tally ds) minus 1) do
 xx := id pick ds;
 x0 := 0 pick xx;
```

```
x1 := 1 pick xx;
cd := id + 1;
yy := cd pick ds;
y0 := 0 pick yy;
y1 := 1 pick yy;
if ec = 0 then
mk := x0 pick mlist;
 if tally mk > 1 then
 g := mk find kck;
 rec := g pick nsn;
 else g := mk find pck;
 rec := g pick sns;endif;
endif;
if x0 = y0 then
ec := 1;
if tally rec = 1 then
rec := rec link '( ' link (x0 pick mlist) link
      ',' link (x1 pick mlist) link ',' link
      (y1 pick mlist) link ',';
else rec := rec link (y1 pick mlist) link ',';
endif;
else rec := (rest rec) link ' )';
writescreen sketch rec;
ec := 0;
endif;
endfor;
writescreen. sketch rec link ' )';
mi := trans (tc > r);
for kc with tell (tally mlist) do
mi := 0 (kc kc) place mi;
endfor;
sd := 1 findall mi;
for di with tell (tally sd) do
xx := di pick sd;
x1 := 0 pick xx;
x2 := 1 pick xx;
g1 := x1 pick mlist;
g2 := x2 pick mlist;
if g2 notin pck then
writescreen. '( ' link sketch g1 link ',' link
             sketch g2 link ')';
else
writescreen. sketch g1 link ' <<--> ' link sketch g2;
endif;
endfor;
)

design is op x(
repeat
```

```
writescreen ' SELECT THE FUNCTION IN THE
             FOLLOWING MENU';
writescreen ' ';
writescreen ' ';
writescreen ' ';
writescreen ' 1. CONCEPTUAL DESIGN';
writescreen ' ';
writescreen ' ';
writescreen ' 2. STRUCTURE REFINMENT';
writescreen ' ';
writescreen ' ';
writescreen ' 3. DESIGN RESULT';
a := read ' ';
case a from
1 : repeat
    writescreen 'SELECT THE FUNCTION FROM
                 THE FOLLOWING MENU';
    writescreen ' ';
    writescreen ' ';
    writescreen ' 1. DATA ENTRY';
    writescreen ' ';
    writescreen ' 2. REEDITING';
    writescreen ' ';
    writescreen ' 3. SEARCH REDUNDANCY';
    writescreen ' ';
    writescreen ' 4. MAPPING BETWEEN KEYS';
    writescreen ' ';
    writescreen ' 5. DISPLAY DESIGN DIAGNOSTIC';
    writescreen ' ';
    writescreen ' 6. EXIT';
    b := read ' ';
    case b from
    1 : writescreen'SELECT THE TYPE OF RECORD';
        writescreen'1. RECORD WITH SIMPLE KEY';
        writescreen' ';
        writescreen'2. RECORD WITH COMPOSITE KEYS';
        bb := read ' ';
        if bb = 2 then check := 1 else check := 0; endif;
        e := entry e;
        attr 0;
        key 0 ;
        num 0
        end
    2 : reedit 0 end
    3 : num 0; search 0 end
    4 : map 0 end
    5 : displays 0 end
    6 : cj := 1 end
    else writescreen 'ERROR ? TRY AGAIN !' endcase;
```

```
    until cj = 1 endrepeat ;
    if uh ~= 1 then gen 0; endif
    end
2 : repeat
    writescreen 'SELECT FROM THE
                 FOLLOWING MENU';
    writescreen ' ';
    writescreen ' ';
    writescreen ' 1. TRANSITIVITY REMOVAL';
    writescreen ' ';
    writescreen ' 2. REDUCING THE NO OF ERs';
    writescreen ' ';
    writescreen ' 3. DISPLAY STRUCTURE DIAGNOSTIC';
    writescreen ' ';
    writescreen ' 4. EXIT';
    v := read ' ';
    case v from
    1 : alt 0; mtrx 0; closure mx; sol 0 end
    2 : reduces tc; frame r mlist  end
    3 : displays 0 end
    4 : gj := 1 end
    else writescreen'ERROR ? TRY AGAIN !' endcase ;
    until gj = 1 endrepeat end
3 : result 0;
    writescreen 'DESIGN COMPLETED';
    jc := 1 end
else writescreen 'ERROR ? TRY AGAIN !' endcase;
until jc = 1 endrepeat;
)
```

VITA

The author was born to Capt. Kai-Ming Chong and his wife Ya-Min Chung in Taipei, Taiwan, the Republic of China, on December 27, 1959. He completed his undergraduate study in Tunghai University, Taichung, Taiwan, and received a Bachelor of Science degree in Industrial Engineering in June 1981. Upon graduation, he was called to military service for two years. After that, he came to the United States for his graduate studies.