8-1983

An Optimal Design of Relational Data Model

Pissamai Amatayakul

Western Michigan University

Follow this and additional works at: http://scholarworks.wmich.edu/masters_theses
Part of the Computer Sciences Commons

Recommended Citation
http://scholarworks.wmich.edu/masters_theses/1566

This Masters Thesis-Open Access is brought to you for free and open access by the Graduate College at ScholarWorks at WMU. It has been accepted for inclusion in Master's Theses by an authorized administrator of ScholarWorks at WMU. For more information, please contact maira.bundza@wmich.edu.
AN OPTIMAL DESIGN OF REALTIONAL DATA MODEL

Pissamai Amatayakul, M.S.
Western Michigan University, 1983

The objective of this study was to propose a design methodology to support a designer in mapping a conceptual model to a logical model. The system was created as a semininteractive software to produce an optimal relational data model for a given database. The software was developed by combining a well known normalization method from a theory of relational data model which was presented by Codd and the ideas of new normal form which were presented recently by many researchers in the field of database design. The complete description of data objects and their associations are initially defined as a set of relations, along with each relation the dependencies are defined as constraints. Afterward a normalized schema is derived by the execution of the software under the consideration of a database designer.
ACKNOWLEDGEMENTS

I am deeply indebted to Dr. Dalia Motzkin for her guidance and encouragement during the course of this study and the hours of struggle and enthusiastic support and for introducing me to the area of database design. I would like to thank Mrs. Donoghue Billie. She spent her time providing me with all the information about the format of the master's thesis. The other person whose kindness I will never forget is my best friend, Mr. Tweeporn Saikasem. He always gave me motivation and assistance to work during the time I was working on this project. I would like to thank him for all the good things he did for me.

Pissamai Amatayakul
INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or “target” for pages apparently lacking from the document photographed is “Missing Page(s)”. If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of “sectioning” the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.
PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark √.

1. Glossy photographs or pages ______
2. Colored illustrations, paper or print ______
3. Photographs with dark background ______
4. Illustrations are poor copy ______
5. Pages with black marks, not original copy ______
6. Print shows through as there is text on both sides of page ______
7. Indistinct, broken or small print on several pages ______
8. Print exceeds margin requirements ______
9. Tightly bound copy with print lost in spine ______
10. Computer printout pages with indistinct print ______
11. Page(s) ______________ lacking when material received, and not available from school or author.
12. Page(s) ______ seem to be missing in numbering only as text follows.
13. Two pages numbered ______________. Text follows.
14. Curling and wrinkled pages ______
15. Other ________________________________

University Microfilms International

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>ii</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>vi</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vii</td>
</tr>
<tr>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>REVIEW OF RELATIONS AND DEPENDENCIES</td>
<td>3</td>
</tr>
<tr>
<td>Functional Dependency</td>
<td>5</td>
</tr>
<tr>
<td>One-to-One Dependency</td>
<td>5</td>
</tr>
<tr>
<td>Multivalued Dependency</td>
<td>6</td>
</tr>
<tr>
<td>DESIGNING STEPS</td>
<td>9</td>
</tr>
<tr>
<td>Expression of Local View</td>
<td>9</td>
</tr>
<tr>
<td>Designing Conceptual Schema</td>
<td>9</td>
</tr>
<tr>
<td>Decomposition into Normal Forms</td>
<td>10</td>
</tr>
<tr>
<td>Adjust the Normal Forms</td>
<td>20</td>
</tr>
<tr>
<td>THE SYSTEM DESCRIPTION</td>
<td>22</td>
</tr>
<tr>
<td>File Description</td>
<td>22</td>
</tr>
<tr>
<td>Running Procedure</td>
<td>31</td>
</tr>
<tr>
<td>Output Documentation</td>
<td>40</td>
</tr>
<tr>
<td>THE DESCRIPTION OF THE ALGORITHMS</td>
<td>42</td>
</tr>
<tr>
<td>Process New Database Schema</td>
<td>43</td>
</tr>
<tr>
<td>Modify Old Database Schema</td>
<td>51</td>
</tr>
<tr>
<td>CONCLUSIONS</td>
<td>53</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

Table

1. STUDENT Relation........................... 3
2. AUTO-ORDER Relation.......................... 5
3. DEPT-EMPLOYEE Relation......................... 6
4. EMPLOYEE Relation............................. 7
5. PATIENT-ADMITANCE Relation.................... 12
7. DEPT-ACCOUNT Relation.......................... 14
8. EMP-CHILD Relation............................. 17
9. EMP-SALARY Relation............................ 17
10. TELEPHONE Relation............................ 18
11. FUND-REQUEST Relation......................... 19
LIST OF FIGURES

Figure

1. The Example of Output Report ................. 89
2. The Example of Error Checking Report ...... 107
INTRODUCTION

In the last few years, the commercial use of databases has increased tremendously. The database has grown in size and complexity. The important problem in designing a system is to integrate all requirements from users' applications to create an appropriate database design. The relational data model introduced by Codd (1970) is widely used in commercial application. Codd (1970) presented the method of normalization to eliminate redundancy and update problems. Relations are decomposed to lower order relations which are simpler and unambiguous. Bernstein (1976) and Beeri, Fagin and Howard (1977) presented the efficient algorithms for synthesizing a relational data model from a given set of functional dependencies. Later Fagin (1977) found another type of dependency which caused some problems of redundancy in a relation. He introduced a definition of multivalued dependency. The further study of this kind of dependency led him to a definition of new normal form which is called fourth normal form. The designing is no longer a trivial problem that can be easily done by a database designer without computer aid because a designer has to keep track of new relations generated by each step of normalization and consider the existence of possible dependencies needed to be removed from relations in order to eliminate redundancy.
The purpose of this study is to provide an efficient tool for a database design. The software was created to normalize a set of schema by using dependencies as constraints and the result is an optimal set of relational data model in a suitable normal form. The normalization is based on the decomposition methods followed by some synthesis. A description of the software is found in section 4 and section 5. In section 4 is the explanation of how to create input files and the following section is the detail of algorithms and the normalization steps. For a better understanding of the detail of the system, the examples of databases are shown in the appendix A.
The relational data model provides a means of describing data, logical associations and relationships of interest to the users of database with its natural structure. The database designer collects data objects, relationships between the data and then defines entities and their associations. The data are organized into a set of relations. Each relation is viewed as a table where each row of the table describes an entity and each column corresponds to an attribute of the entity. For each attribute there is a set of possible associated values called the domain of that attribute. A relation is denoted by \( R(A_1,A_2,...,A_n) \) where \( A_1, A_2,..., A_n \) are called the attributes of the relation \( R \). Each attribute \( A_i \) uniquely associated with a domain which is a set of all possible values for that attribute. The relation \( R \) is defined as the subset of the cartesian product of its domains \( A_1 \times A_2 \times ... \times A_n \).

Table 1

<table>
<thead>
<tr>
<th>S-ID</th>
<th>S-Name</th>
<th>Address</th>
<th>Sex</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>900-01-1349</td>
<td>Mary</td>
<td>200 Spindler</td>
<td>F</td>
<td>2</td>
</tr>
<tr>
<td>300-05-8648</td>
<td>Tom</td>
<td>100 French</td>
<td>M</td>
<td>3</td>
</tr>
<tr>
<td>468-30-5601</td>
<td>Edward</td>
<td>251 David</td>
<td>M</td>
<td>4</td>
</tr>
</tbody>
</table>
This relation is called a relation with degree 5. Each row represents an n-tuple. The ordering of rows is unimportant and all rows are distinct. It is required in that the domain names should be qualified by a distinctive role of a domain in a relation. A relation with high degree is not practical. It is important to make a distinction between a relation scheme and a relation. The scheme is a representation of the relation, it is composed of a set of attributes, the domains of the attributes and the primary key. A relation is pictured as a two dimensional table, each row contains one value for each attribute. This paper is not concerned with the implementation of the relations.

A key for a relation is an attribute or a set of attributes which uniquely identifies a tuple of that relation. No two tuples can have the same key value. It may be that more than one set of attributes are possible keys. They are called candidate keys but only one of them is chosen to be a primary key of a relation. For example "S-ID" is selected to be a primary key for a STUDENT relation.

By observing a set of attributes, it is found that there are different types of dependencies which are described below.
Functional Dependency

When the value of attribute B is determined by the value of another attribute A, it is said that B is functionally dependent on A. It can be written by using the notation $A \rightarrow\rightarrow B$.

Table 2
AUTO-ORDER Relation

<table>
<thead>
<tr>
<th>DEALER</th>
<th>AUTO</th>
<th>STATE</th>
<th>QTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Cardillac</td>
<td>MI</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>Cardillac</td>
<td>IN</td>
<td>20</td>
</tr>
<tr>
<td>A</td>
<td>Ford</td>
<td>MI</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>Ford</td>
<td>WI</td>
<td>10</td>
</tr>
</tbody>
</table>

There are two functional dependencies in this relation.

FD1 DEALER, AUTO $\rightarrow\rightarrow$ QTY
FD2 DEALER $\rightarrow\rightarrow$ STATE

One-to-One Dependency

When each attribute value in a domain A is paired with a unique attribute value in domain B vice versa, it is said that A is one-to-one dependency to B and is denoted by $A <--\rightarrow B$. In this case a designer should determine which is a primary attribute.
Table 3
DEPT-EMPLOYEE Relation

<table>
<thead>
<tr>
<th>DEPARTMENT</th>
<th>MANAGER</th>
<th>TOTAL-EMPLOYEE</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marketing</td>
<td>J.B.</td>
<td>20</td>
<td>B3-102</td>
</tr>
<tr>
<td>Sys. Service</td>
<td>M. Thomas</td>
<td>34</td>
<td>B3-200</td>
</tr>
<tr>
<td>Personel</td>
<td>W. Spence</td>
<td>27</td>
<td>B1-104</td>
</tr>
</tbody>
</table>

It is assumed that each department has a unique manager. The department name implies a manager of the department and a manager name implies which department a manager belongs to. The dependencies found in this relation are

D 1. DEPARTMENT \(\leftarrow\) MANAGER
D 2. DEPARTMENT \(\rightarrow\) TOTAL-EMPLOYEE
D 3. DEPARTMENT \(\rightarrow\) LOCATION
D 4. MANAGER \(\rightarrow\) TOTAL-EMPLOYEE
D 5. MANAGER \(\rightarrow\) LOCATION

In the system when a designer determines which is a primary attribute, in a one-to-one dependency. A designer should put a primary attribute on the left hand side of the line. In this example DEPARTMENT is a primary attribute.

Multivalued Dependency

Let \(R(X,Y,Z)\) be a relation on a set of attributes that contains \(X, Y,\) and \(Z\). If set \(Y\) depends on the value of an
attribute in set X but there is more than one Y-value for a
given X-value, and if each combination of X and Z will yield
the same set of Y's then it is said that Y is multivalued
depends on X. The notation is $X \longrightarrow Y$.

Table 4

<table>
<thead>
<tr>
<th>EMP-SS</th>
<th>CHILD-NAME</th>
<th>SALARY</th>
<th>YEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Tom</td>
<td>500</td>
<td>1979</td>
</tr>
<tr>
<td>01</td>
<td>Mary</td>
<td>500</td>
<td>1979</td>
</tr>
<tr>
<td>01</td>
<td>Tom</td>
<td>800</td>
<td>1980</td>
</tr>
<tr>
<td>01</td>
<td>Mary</td>
<td>800</td>
<td>1980</td>
</tr>
</tbody>
</table>

A set of child name values is dependent on a given EMP-SS.
It is not dependent on other attribute values in a relation.
Some sets of child names is repeated with each combination of
the value of EMP-SS with SALARY and YEAR.

Many researchers in a database design have studied the
important roles of functional dependencies in a relation.
Sometimes a relation contains the associations between non
key attributes, or the associations between a subset of key
attributes and non key attributes. Codd introduced the
definition of second and third normal form to correct the
problems of anomalies and redundancies. Afterward it has been observed that other anomalous situations may occur because the concept of functional dependency is not general enough to describe the users' view of their applications. There are some dependency types which are not functional dependency and may lead to redundant problems when data is stored in the database. From the table 4, a set of attribute values EMP-SS, SALARY and YEAR must be repeated for every child name. If the employee has ten children, the redundancy is easily seen. The study of this dependency and its influence on a relational schema design were presented along with a definition of fourth normal form to improve the performance of third normal form. The following section will discuss the approach of normal form to the designing of relational data model.
DESIGNING STEPS

The design of a relational data model is developed through a sequence of successive improvement steps as follow:

Expression of Local View

The end users of the database express their views of the applications in terms of attributes, their associations and their constraints. The data should come from the users' view because the users are the persons who understand their work and they can define the constraints and the requirements for their applications correctly than any other people. The data model will be a representation of a real world situation for a database.

Designing Conceptual Schema

The designer integrates all information from the end users' views and produces a first model of the database in the form of a set of relations. When data attributes are grouped together, the problems come from the fact that some certain attributes completely determine other attributes in a relation. In order to minimize redundancy and update problems, the designer has to produce a set of constraints.
which are given by the end users and then to define them as a set of data dependencies. The method of how to declare the consistent and the complete dependencies for a set of relations are not proposed in this paper.

Decomposition into Normal Forms

A normal form refers to a class of relational schema which obeys some set of rules. The set of relations which is initially defined from the users' views is often not in a suitable normal form. Some relations contain redundant attributes. Some of them have the anomaly problems resulting from partial functional dependencies or from transitive functional dependencies which remain in relations. Codd presented the method of normalization to eliminate anomalies and ambiguous relationships between attributes in a relation. By using the information from a set of dependencies one can decompose the initial set of relations into lower order relations which satisfy a suitable normal form. The decomposition algorithm described below is a recursive procedure. At each step, each relation is decomposed into a pair of subprojections by using the constraints from the given dependencies which exist in that relation. Then these projections are decomposed in turn and the process continues until a set of normalized relations is obtained. The objectives of the decomposition are the
following:

1. Decomposing complex relations into simple well defined primitives
2. Preserving information
3. Minimizing redundancy

The decomposition techniques are discussed for each type of data dependencies.

**Decomposition into Second Normal Form**

When a relation contains partial dependencies on the key, it is said that some sets of non key attributes are not dependent on the whole keys of a relation. Under this condition the anomalies can be removed by decomposing the relation in such a way that in new relations, no attributes are dependent upon a sub set of candidate keys. When a relation is constructed due to such functional dependency, the set of left hand side attributes becomes a set of key attributes for a new relation. The other relation is what remains from the original relation after a set of attributes which are partially dependent on the key are removed.
Table 5
PATIENT-ADMITANCE Relation

<table>
<thead>
<tr>
<th>P-ID</th>
<th>DATE-ADMIT</th>
<th>P-ADDR</th>
<th>SECTION-ADMIT</th>
<th>SYMTPMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>2/13/82</td>
<td>130</td>
<td>Ferndale MI</td>
<td>49007 02</td>
</tr>
<tr>
<td>001</td>
<td>5/10/82</td>
<td>130</td>
<td>Ferndale MI</td>
<td>49007 02</td>
</tr>
<tr>
<td>002</td>
<td>5/10/82</td>
<td>40</td>
<td>West Main MI</td>
<td>49007 05</td>
</tr>
<tr>
<td>610</td>
<td>4/11/83</td>
<td>210</td>
<td>W.Mich MI</td>
<td>49008 11</td>
</tr>
<tr>
<td>001</td>
<td>9/10/83</td>
<td>130</td>
<td>Ferndale MI</td>
<td>49007 08</td>
</tr>
</tbody>
</table>

(F-ID,DATE-ADMIT,P-ADDR,SECTION-ADMIT,SYMTPMS) 5.1

FD 1. P-ID -----> P-ADDR

In this example, the P-ADDR is partially dependent on the set of keys. The P-ADDR is not dependent on DATE-ADMIT which is a part of the key. If any patients change addresses, the update program must go through each record of a given P-ID to change the name on multiple records because it is possible that one patient has more than one visit in the hospital. By the definition of second normal form, the original relation in 5.1 is decomposed into new relations in 5.2 and 5.3.

(P-ID, P-ADDR) 5.2

(P-ID,DATE-ADMIT,SECTION-ADMIT,SYMTPMS) 5.3
Decomposition into Third Normal Form

Some relations which are in second normal form still contain another type of dependency which is called transitive dependency. This kind of dependency is a source of anomalies and should be removed from a relation. The third normal form definition by Codd is a relation in second normal form which does not have transitive dependencies. An example of a relation schema which contains a transitive dependency is shown below:

(P-ID, OPERATE-DATE, DOCTOR-ID, SPECIALIZE-IN, SYMTOMS, SIDE-EFFECT) 6.1

The functional dependencies are

FD 1. P-ID, OPERATE-DATE ----> DOCTOR-ID
FD 2. DOCTOR-ID ----> SPECIALIZE-IN

FD 2. is a transitive functional dependency. The problem is that when a new doctor is added along with his specialization but at that time he/she does not have any patients under operations. The set of keys cannot have an undefined value. It was suggested that the relationship between DOCTOR-ID and SPECIALIZE-IN should not have been in this relation. When the relationship between DOCTOR-ID and SPECIALIZE-IN is projected from the original relation the designer may extend the new schema in the future. He/she may want to add more attributes which are related to DOCTOR-ID.
After decomposing the original relation, the new relations in third normal form are

(\text{DOCTOR-ID, SPECIALIZE-IN}) \hspace{1cm} 6.2

(\text{P-ID, OPERATE-DATE, DOCTOR-ID, SYMTPMS, SIDE-EFFECT}) \hspace{1cm} 6.3

\textbf{One-to-One Dependency}

When two sets of attributes have one-to-one dependency to each other, a designer then makes a decision to select which one should be a set of primary attributes. If any relations contain one-to-one dependencies, a set of non primary attributes should be eliminated from a relation to reduce redundancy, except for candidate keys. (This is a special case of transitive dependency, it is used in the algorithm to obtain an optimal model.)

\begin{table}
\centering
\caption{DEPT-ACCOUNT Relation}
\begin{tabular}{lll}
DEPT & MG & ACC-NUMBER \\
Personel & Gale J.A. & 010 \\
Personel & Gale J.A. & 050 \\
Personel & Gale J.A. & 065 \\
Accounting & Paul W. & 065 \\
Accounting & Paul W. & 011 \\
Inventory & James S. & 010 \\
Inventory & James S. & 050 \\
\end{tabular}
\end{table}

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
It is assumed that no more than one manager belongs to the same department. Each department may have several accounts. The key attributes can be either DEPT and ACC-NUMBER or MG and ACC-NUMBER because DEPT and MG uniquely identify each other. The designer defines one-to-one dependency and puts the primary attribute in the left hand side. By using the property of one-to-one dependency to decompose the original relation, the new relations are in 7.2 and 7.3

If the manager of a particular department has to be changed, it is easier to go through relation 7.2, searching for the department name in which there is only one unique record in the file of 7.2 schema.

Decomposition into Fourth Normal Form

From the objective of decomposition the elimination of anomalies is not adequate for designing a database schema. The other objective which is also important is to present the complete reliability of data. The third normal form relations do not ensure completeness of information. By the study of multivalued dependency between attributes several
researchers introduced the definition of fourth normal form decomposition which is a generalization of third normal form. Multivalued dependencies provide a necessary and sufficient condition for a relation to be decomposable into two relations without loss of information. A relation which is in third normal form and does not contain any multivalued dependencies is said to be in fourth normal form. The relation shown in table 4 is in third normal form but the redundancy is easily seen when CHILD attribute values are put in the relation. The other attributes have to be repeated for each child name. There is a multivalued dependency between EMPLOYEE and CHILD.

(EMPLOYEE CHILD SALARY YEAR) 8.1

MD 1. EMPLOYEE -->> CHILD

This redundancy is removed by using a similar rule as functional dependencies. The attribute CHILD is removed from the relation and the new relation is formed from the MD 1., usually the left hand side attributes become a set of key attributes but in this case the CHILD attribute is a part of a set of keys in the original relation so it also becomes a part of keys for the new relation. After decomposing 8.1. The new relations are

(EMPLOYEE, CHILD) 8.2

(EMPLOYEE, SALARY,YEAR) 8.3

The new relations perform a better schema design beyond the
idea of third normal form. When any CHILD attribute value is updated for a particular employee, the program easily searches in 8.2 and updates the desired records without going through unnecessary records in 8.1. The new relations also solve the problem of assigning dummy values for the attributes. If an employee does not have any children by the first time of his employment, the dummy value is not necessary to assign.

Table 8  Table 9

<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th>CHILD</th>
<th>EMPLOYEE</th>
<th>SALARY</th>
<th>YEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Mary</td>
<td>John</td>
<td>1000</td>
<td>1979</td>
</tr>
<tr>
<td>John</td>
<td>Tom</td>
<td>Joan</td>
<td>2000</td>
<td>1980</td>
</tr>
<tr>
<td>John</td>
<td>Jane</td>
<td>Mark</td>
<td>3000</td>
<td>1981</td>
</tr>
<tr>
<td>Mark</td>
<td>Tony</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The employee "Mark" got hired in 1981 and at that time he did not have any children. After he worked for a while, he got a child named Tony. This new information is inserted directly in EMPLOYEE-CHILD. In another case, when a new salary for a particular employee is inserted, the update program does not have to repeat the same information of the new salary for every employee's children.
Decomposition with respect to the Representation Principle

Some relations cannot be decomposed by using the functional dependency rules or multivalued dependency rules to obtain better schemata. There is a different way to decompose a relation when a set of non key attributes determines a subset of key attributes.

Table 10

TELEPHONE Relation

<table>
<thead>
<tr>
<th>AREA-CODE</th>
<th>NUMBER</th>
<th>PLACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>616</td>
<td>345-1023</td>
<td>Kalamazoo</td>
</tr>
<tr>
<td>517</td>
<td>450-3454</td>
<td>Lansing</td>
</tr>
<tr>
<td>312</td>
<td>782-4567</td>
<td>Chicago</td>
</tr>
<tr>
<td>616</td>
<td>923-8780</td>
<td>Battle Creek</td>
</tr>
</tbody>
</table>

(AREA-CODE, NUMBER, PLACE) 10.1

FD 1. AREA-CODE, NUMBER ----> PLACE

FD 2. PLACE ----> AREA-CODE

By the normal decomposition, the new relations are

(AREA-CODE, PLACE) 10.2

(NUMBER, PLACE) 10.3

The second relation (10.3) is not valid for a given constraint because NUMBER and PLACE do not have relationship
to each other. The NUMBER cannot determine the value of
PLACE. The better way to decompose is proposed by a concept
of new normal form, the original relation should be replaced
by

\[(\text{PLACE, AREA-CODE})\] 10.4
\[(\text{AREA-CODE, NUMBER, PLACE})\] 10.5

Consider the next example shown in table 11.

Table 11
FUND-REQUEST Relation

<table>
<thead>
<tr>
<th>FUND-NO</th>
<th>ASKED-BY</th>
<th>AMT-REQUEST</th>
<th>PURPOSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>K-COMMU</td>
<td>1000</td>
<td>CHILDCARE</td>
</tr>
<tr>
<td>002</td>
<td>K-COMMU</td>
<td>800</td>
<td>DRUG-CONTROL</td>
</tr>
<tr>
<td>005</td>
<td>K-COMMU</td>
<td>500</td>
<td>BIRTH-CONTROL</td>
</tr>
<tr>
<td>001</td>
<td>A-COMMU</td>
<td>1000</td>
<td>CHILDCARE</td>
</tr>
<tr>
<td>001</td>
<td>K-COMMU</td>
<td>1500</td>
<td>CHILDAHUSE</td>
</tr>
</tbody>
</table>

\[(\text{FUND-NO,ASKED-BY, PURPOSE, AMT-REQUEST})\] 11.1

FD 1. FUND-NO, ASKED-BY --------> AMT-REQUEST
FD 2. FUND-NO, ASKED-BY --------> PURPOSE
FD 3. PURPOSE --------> FUND-NO

It is assumed that each fund is to serve particular
purpose, the organizations can ask for any fund for each
purpose. The PURPOSE can be used to determine particular
types of FUND-NO. To preserve this information the results from the decomposition are

(FUND-NO, ASKED-BY, AMT-REQUEST, PURPOSE) 11.2
(FUNDO, FUND-NO) 11.3

The decomposition will ensure the preservation of both dependency structures and content of the original relation. The redundancy is minimized by each step of decomposition. The result is an optimal set of relations. The normalization can be viewed as processes for eliminating redundancy from the definition of a database schema while preserving the complete relatability of the data. It is obvious that the quality of the schema produced by the step of normalization depends on the correctness and completeness with which a designer initially defines all dependencies. The normalization works under the assumption that the initial definition of the data objects and dependencies are correct.

Adjust the Normal Forms

After a set of relations is in an optimal form, the next approach is to let a designer get involved in the loop of normalization. A designer will have a chance to review and modify the result schema. If a designer finds any dependencies which were neglected at the beginning design step, the designer can add them. Some multivalued
dependencies are not valid in a relation but when a relation is decomposed they are valid in the new relations. A designer can define these kinds of dependencies to each relation and the normalization will reprocess from the first step to the last step by using new dependencies to decompose relations. The loop of normalization is repeated until a designer does not define any more dependencies. There are two reasons of this approach. The first reason is that most people do not want to accept a totally automated results from the computer. In this approach a designer is a person who decides to stop the process. The final result comes from his/her consideration. The second reason is that if some dependencies were omitted at the beginning, they can be added to the appropriate relations. The process of normalization can continue without going back to the initial step in order to add the missing dependencies.
THE SYSTEM DESCRIPTION

The system is designed to manipulate a given global conceptual database schema to an optimal logical data model. The method is based on the theory of relational data model. A designer collects all information regarding all entities, all associations and a sufficient set of dependencies. The information is organized into two input files. The first file contains all relations, their attributes and primary keys. The second file contains the set of needed dependencies which are defined over each relation in the first file. The format of the input files is described in the following section.

File Description

Relations File

This file is a collection of a set of relations which are in first normal form. Each relation is defined over a set of attributes. A set of primary key attributes is determined for each relation. It is assumed that the relations file is a complete representation of an enterprise prepared by a database designer.
How to Create a Relation File

The file is created by using text editor "SOS". The user may select any name for the relations file. The following steps are required.

1. Enter a relation name.

2. Enter each attribute of the primary key line by line.

3. Enter each non key attribute line by line.

4. Enter the end of relation by typing "END RELATION"

5. Go back to step 1 to enter the next relation.

6. When all relations have been entered, type "EOF".

At the beginning of each line is a preceding symbol used to indicate the meaning of a successive name. There are four different symbols.

1. " /" precedes a relation name.

2. " #" precedes a primary key attribute name.

3. " *" precedes a non key attribute name.
4. "$" precedes the end of relation or the end of file.

5. "%" precedes a comment line.

Suppose there is a set of two relations as follows:

\begin{verbatim}
R1(P-ID, ADMIT-DATE, P-NAME, ADDR, SS-NO,
   SECTION-ADMIT, SYMTOMS, TREATMENT, DOCTOR)
R2(P-ID, OP-DATE, PART-OPERATE, DOCTOR, LICENSE-NO,
   SPECIALIZE-IN, WHERE-TAKEN, RESULT, SIDE-EFFECT)
\end{verbatim}

"R1" and "R2" are the relation names.

The underlined attributes indicate the primary keys for the relations.

The rest of the attributes are non key attributes.

The format of a relations file is

\begin{verbatim}
col. 12345678901234567890.....
/ R1
# P-ID
# ADMIT-DATE
* P-NAME
* ADDR
* SS-NO
* SECTION-ADMIT
* SYMTOMS
* TREATMENT
* DOCTOR
$ END RELATION
/ R2
# P-ID
...
$ END RELATION
$ EOF
\end{verbatim}
The first column is a blank.
The second column is a preceeding symbol.
The third column is a blank.
Starting at the fourth column, one of this following is stored, a name of an attribute, the end of a relation, the end of file. The lengths of relation names and attribute names are limited to 30 characters.

Dependencies File

The dependencies file is a collection of a set of dependencies which represents a set of constraints for a database. There are three types of dependencies which are explained in the previous section. They are denoted by the three different symbols.

1. "----->" denotes a functional dependency.
2. "<-----" denotes a one-to-one dependency.
3. "----->" denotes a multivalued dependency.

The dependencies must be defined for each relation separately. A designer should be careful in the problem of consistency and integrity. It is not necessary to define dependencies between a set of key attributes and each of non key attribute. The system knows that each attribute in a relation is functionally dependent on the key.
R1(P-ID, ADMIT-DATE, P-NAME, ADDR, SS-NO, SECTION-ADMIT, SYM-TO-M, TREATMENT, DOCTOR-ID)

LEFT ATTR. DEPENDENCY TYPE RIGHT ATTR.
P-ID <-----> P-NAME, ADDR
P-ID <-----> SS-NO

R2(P-ID, OP-DATE, PART-OPERATE, DOCTOR, LICENSE-NO, SPECIALIZE-IN, WHERE-TAKEN, RESULT, SIDE-EFFECT)

LEFT ATTR. DEPENDENCY TYPE RIGHT ATTR.
DOCTOR-ID <------> LICENSE-NO
DOCTOR-ID ----------> SPECIALIZE-IN

How to Create the Dependencies File

The dependencies file is created by using the text editor "SOS". The user may select any name for the dependencies file. Enter dependencies for each relation corresponding to a set of relations in a relations file. The following steps are required.

1. Enter a relation name.

2. Enter a left attribute name line by line.

3. Enter a right attribute name line by line, or enter ";" to separate attributes which are one-to-one dependency.

4. Enter a dependency type. It can be "<----->" or "<----->" or "----->".
5. Go to step 2 to enter another set of dependencies until there is no more dependency for this relation then enter the end of relation by typing "END RELATION".

6. Go to step 1 to enter next relation name.

7. When all relations have been entered, type "EOF".

At the beginning of each line a preceding symbol is used to indicate the meaning of the following name. There are 7 different symbols.

1. " /" precedes a relation name.

2. "L*" precedes a left hand side attribute.

3. "R*" precedes a right hand side attribute.

4. " ;" precedes a blank which is used to separate sets of attributes with one-to-one dependency. In a set of attributes which are 1:1, the left attribute is interpreted as the primary attributes.

5. " $" precedes the end of relation and the end of file.

6. The preceding symbol is separated from the attribute name by a blank.

7. " %" precedes a comment line.
The format of a dependencies file is

```
col. 12345678901234567890.....

/ R1
L* P-ID
;
R* P-NAME
R* ADDR
;
R* SS-NO
<---->
$ END RELATION
/ R2
L* DOCTOR
;
R* LICENSE-NO
<---->
L* DOCTOR
R* SPECIALIZE-IN
----->
$ END RELATION
$ EOF
```

The relation names which are preceded by "/" must be the same as they appear in the relations file. The symbol ";" in the first column indicates the one to one dependency between a set of left attributes and a set of right attributes. If the set of left attributes is one to one to another set of right attributes, the user does not have to type the set of left attributes again. The symbol ";;" will imply the previous set of the left attributes. At the end of defining one to one dependencies is the symbol "<---->". In case the left attributes also define another set of right attributes whose dependency is not one to one dependency, the user must type the set of left attributes again.
Names File

At the beginning step of the execution, the program converts the input files to random files. The relations file is converted into the random relations file. The dependencies file is converted into the random dependencies file. After the end of normalization, the program also generates the output files which are the optimal relations file and the final dependencies file. The names of these four files must be supplied by the user. The user enters these four file names by using SOS to create the file which is called the names file. The names file will have four records as follow:

The first record is the name of the random relations file.

The second record is the name of the random dependencies file.

The third record is the name of the output optimal relations file.

The fourth record is the name of the final dependencies file.
Each name must be exactly nine characters. Blanks are needed to fill in case the length of file name is not nine characters.

For example:

The name of random relations file is "RELAT.INP".
The name of random dependencies file is "FUNCT.INP".
The name of optimal relations file is "OPTMAL.OUT".
The name of final dependencies file is "DEPEND.OUT".

The format of a names file is

col. 1234567890...

RELAT INP
FUNCT INP
OPTMAL OUT
DEPEN DOUT

The first record and the second record will be the names of the input files after the system converted the original relations file and the original dependencies file into random files. The third and fourth record will be the names of the output files after the system normalize all relations into an optimal data model. These output files are random files.

The names file will be converted into a random file but the system will generate the value of id to the random names file automatically. It will be used to refer to a particular database schema in the future.
Running Procedure

A user runs the system in monitor mode, using the command "RUN DESIGN" or using the command "EX
MAIN.CBL,CONVRT.CBL,ERR.CBL,COMB.CBL,CPREES.CBL". The system provides two options for a designer. One is to start a new
database design. The other option is to modify an old database design. Enter the code "1" to start a new database
design. Enter the code "2" to get into a desired database
design.

The messages are:

A DESIGN OF OPTIMAL RELATIONAL DATA MODEL

THE PROGRAM PROVIDES TWO OPTIONS FOR THE USER

1. PROCESS NEW DATABASE DESIGN

2. MODIFY YOUR OLD DATABASE

PLEASE ENTER THE CODE YOU DESIRED ("1" OR "2") :

Process New Database Schema

The process of a new database design can be divided into
7 steps.

1. Enter input file names which have been created by
using "SOS".

The messages are:

ENTER THE FILE NAMES YOU HAVE CREATED BY SOS
1. RELATION FILE =
2. FUNCTION FILE =
3. NAMES FILE =

If a file name is not valid in a user's area, the system gives a warning message and allows the user to enter a new file name.

The messages are:

** THIS FILE IS NOT IN YOUR AREA

ENTER NEW NAME =

In this step new random files are generated from the original three input files.

The messages are:

STEP I. CONVERT THE ORIGINAL FILES TO RANDOM FILES

** END OF STEP I.

2. Check errors for attribute names in a dependencies file corresponding to attribute names in a relation files.

The purpose of this step is to find some mistakes when typing attribute names.

The messages are:

STEP II. CHECK ERROR ATTRIBUTES IN DEPENDENCIES FILE

ENTER ERROR REPORT FILE =

The types of error are classified into 2 cases.

Case 1. Any left hand attributes are not valid in relations.
Case 2. Any right hand attributes are not valid in relations.

A user enters the error report file. If there are such mistakes, the error messages are printed in the error report. The execution halts. A designer should review both the original relations file and the original dependencies file together. The random files which are generated by the system from step 1 should be deleted before a user runs the system.

The messages are:

** END OF STEP II.

** THE ORIGINAL FILES SHOULD BE REVIEW

** YOU MUST GO BACK TO START AT THE BEGINNING

** CANNOT CONTINUE THE PROCESS ...

If there is no error, the next step is processed.

The messages are:

** END OF STEP II.

** NO ERRORS FOUND AT THIS TIME

3. The system begins the normalization steps with the second normal form, third normal form and fourth normal form respectively.

The messages are:

** NORMALIZED IN SECOND NORMAL FORM

!! PLEASE WAIT FOR A WHILE. !

!! THE PROGRAM IS WORKING FOR THIRD NORMAL FORM. !!

** NORMALIZED in FOURTH NORMAL FORM
4. The normalized relational data model is displayed on the screen. A relation name, a set of key attributes and a set of non key attributes are displayed respectively. The system gives an option for a user to change a relation name. The messages are:

** DO YOU WANT TO CHANGE A RELATION NAME ? (Y/N) :

Type "Y" to enter new relation name. Type "N" if a user satisfies a name given by the system. Any other characters are rejected and a user has to enter a new code.

5. After the last relation is displayed, a user has an option to look over the set of relations again.

The messages are:

** DO YOU WANT TO LOOK OVER AGAIN ? (Y/N) :

Type either "Y" or "N" otherwise a user will get a warning message and a user has to enter a new code.

6. A user decides to stop or continue the normalization.

The messages are:

** DO YOU WANT TO DEFINE MORE DEPENDENCIES ? (Y/N) :

If a user type "Y" then each relation is displayed on the screen. The system waits for a user to enter additional dependencies.

The message is:

STEP IV. A USER DEFINES MORE DEPENDENCIES
A relation name, a set of key attributes and non key attributes are displayed.

RELATION NAME = R1
KEY ATTRIBUTE : P-ID
ADMIT-DATE
NON KEY ATTRIBUTE : P-NAME
P-ADDR
SECTION-ADMIT
SYMSTOMS
TREATMENT
DOCTOR

** ANY DEPENDENCIES FOR THIS RELATION? (Y/N) :**

Suppose a designer did not define a one-to-one dependency between P-ID <------> P-NAME, P-ADDR, he/she can enter it at this time.

The messages are:

ENTER LEFT ATTRIBUTES (TYPE "$" TO STOP)
: P-ID
: $

ENTER RIGHT ATTRIBUTES (TYPE "$" TO STOP)
: P-NAME
: P-ADDR
: $

ENTER DEPENDENCY TYPE : <------>

** DO YOU WANT TO CANCEL? (Y/N) :**

If a user makes mistakes or does not want to put this dependency, type the code "Y" to cancel it. If he/she does not want to cancel type the code "N". While a user enters each attribute name the system will check that a name of the attribute is the same as it appears in that relation.
If the name does not match to any attribute names in that relation, the system will show the error messages and let a user enter a new attribute name. The messages are:

** THE ATTRIBUTE DOES NOT EXIST IN THIS RELATION

** PLEASE ENTER NEW ATTRIBUTE NAME

: enter a new name

The type of dependency is also checked by the system. It can be "<---->" or "<---->" or "<--->". If a wrong symbol is entered, the system will give warning messages and ask a user to enter it again. The messages are:

** The dependent type can be

1. "<---->
2. "<---->
3. "<--->"

ENTER DEPENDENCY TYPE AGAIN:

After each relation has been considered to define some dependencies, the process of normalization starts again at the second normal form through the fourth normal form.

7. At the end of the execution, the output optimal relational data model is printed into a print file. A user enters an output file name. The messages are:

ENTER THE REPORT FILE NAME =
Modify Old Database Schema

In case a designer wants to modify a data model by adding some dependencies to any relations or adding more relations to the schema. He/she enters the code "2" at the beginning. After that enter the names file to get to the database schema which he/she wants to modify. The name of names file was given by the system from the first execution of that database schema.

The messages are:

ENTER NAME-FILE =

If the file name is not in a user area the system gives a warning message and waits for a new file name.

In this part, the program gives a user 2 options to modify a database schema. The schema can be modified by off-line or on-line.

The messages are:

THIS MODULE HAS TWO OPTIONS AS FOLLOW:

1. MODIFY THE DATABASE SCHEMA BY OFF-LINE

2. MODIFY THE DATABASE SCHEMA BY ON-LINE

PLEASE ENTER THE CODE YOU DESIRED ("1" OR "2"):
1. OFF-LINE Modification. The purpose of this option is to let a user modify the schema at his/her own pace without computer prompts. The program will generate the final relations file and the final dependencies file from the most recent output files. These two files are sequential files so a user can make any changes as he/she wants to by using SOS. A user can add not only dependencies for each relation but also new relations along with their dependencies into those files. A user can expand the schema any time without going back to enter all original relations and dependencies again. The program will generate the file names for those new files and then exit. When a user is ready to run the program on the modified files, a user has to create a new name file for the next execution and then uses the command "RUN DESIGN" or uses the command "EX MAIN.CBL, CONVRT.CBL, ERR.CBL, COMB.CBL, CPRESS.CBL." At the beginning step, a user is to choose the option "1" as when processing a new database.

The messages are:

** TO MODIFY THE OLD DATABASE SCHEMA BY OFF-LINE
** THE FOLLOWING FILE NAMES CAN BE MODIFIED BY USING "SOS"

THE RELATIONS FILE NAME IS xxxxxxx.xxx
THE FUNCTION FILE NAME IS xxxxxxx.xxx

** THE PROGRAM WILL EXIT TO LET YOU MODIFY THE FILES.
**PLEASE REMEMBER,**

YOU ARE TO CHOOSE THE OPTION "1" TO RUN
THE MODIFIED INPUT FILES.

YOU ALSO HAVE TO CREATE A NAME-FILE FOR
YOUR NEXT EXECUTION.

THE NAMES OF THOSE TWO FILES CAN BE RENAMED BY YOU
LATER.

**END OF EXECUTION**

********************

2. **ON-LINE Modification.** This option is provided in
case a user prefers to define more dependencies directly by
the system. It does not let a user add new relations to the
schema. The same routines as described in the step 6 through
step 7 are performed.

The examples of JOBLOG are shown for each step of the
execution. The first example is the first run of the design
from step 1 through step 7 of the process of new database
schema. The second example shows the modification of the
schema by on-line. The third example shows the modification
of schema by off-line.

The examples are in the appendix F. They are in the
sequence as described above.

**Output Documentation**

The output is the report of an optimal set of relational data
model. A set of underlined attributes is a set of primary
keys for a relation. At the end part of the report is a remark to show the list of one-to-one dependencies which were defined by a designer. The example of the output report is shown in figure 1.

The report of error file is provided to explain the type of error and the list of error attribute names in a dependency file. The example of the relation file and the dependency file show how the errors might occur in both files.

RELATION FILE                DEPENDENCY FILE
/ R1                          / R1
# P-ID                        L* P-ID
# ADMIT-DATE                  ;
* P-NAME                      R* P-NAME
* DOCTOR                      R* ADDR
* P-ADDR                      ;
* P-SS                        R* P-SS
* SECTION-Admit               $ END RELATION
* SYMPTOMS                    / R2
* TREATMENT                   R* DOCTOR
* END RELATION                R* SPECIALIZE-IN
/ R2                          $ EOF
# P-ID                        ---->
# OPERATION-DATE              L* DOCTOR
* SIDE-EFFECT                 ;
* RESULT                      R* LICENSE-NO
* WHERE-TAKEN                 $ END RELATION
* PART-Operate                $ EOF
* DOCTOR                      $ EOF
* LICENSE-NO                  $ EOF
* SPECIALIZE-IN
$ END RELATION
$ EOF

The attribute "LICENSE-NO" of the second relation in the relation file should be "LICENSE-NO". The attribute "DOTOR"
of the second relation in the dependency file should be "DOCTOR". The attribute "ADDR" of the first relation in the dependency file should be "P-ADDR". The error messages are shown for each relation by specifying particular dependencies which the error attributes belong to. In the figure 2 is the report of error checking.
A DESCRIPTION OF THE ALGORITHMS

The project is designed to normalize a relational data model for building a database schema. The structure of the project consists of one main program and four subprograms which are called directly from the main program. The main program takes care of all steps of normalization. It also provides an option for a user to be able to modify an old data model if a user found that some additional dependencies needed to be defined. The method of normalization is based on Codd’s (1970) normal form. A relation is normalized to second, third and fourth normal forms. To remove unnecessary redundancy from relation schemes, one to one dependencies between attributes are considered Some attributes will be automatically replaced by primary attributes. Finally if a database designer finds that there are still some dependencies in a relation, he/she has an option to enter more dependencies and the program will resume the process of normalization again. On the other hand, the program provides the off-line modification for a designer who wants to add more relations and other dependencies inorder to expand the schema or remodify the schema in the future. It is very important to have a good understanding in the theory of normalization before going through the details of the programs.
Process New Database Schema

1. Create random files by calling "CONVRT.CBL".

The original input files are created to be random files so that when relations are decomposed the new relations can be inserted to the file and the old relations can be deleted by putting "%" in the front field of their records. The dependencies which are used to decompose the relation will be deleted in the same way. It helps a designer keep track of the normalization steps if there are some errors during the execution. The detail of this subprogram is explained in the appendix C.

2. Check error attributes in the dependencies file.

By the algorithm, it is assumed that the attribute names in the relations file are correct. Each attribute name in the dependency file is checked with the attribute names in the relations file. The detail of this subprogram is explained in the appendix D.

While not end relations

Read a relation into a relation array which is separated into the array of key attributes, the array of non key attributes.

Search for a set of dependencies which has the same relation name as the name in the relation array.

If it is found
Read a set of dependencies into the function array which is separated into the array of left attributes, the array of right attributes and dependent type.

Initialize a dependency index to 0.

Repeat

Increase a dependency index by 1.

Get each left attribute (L-ATTR) to compare with each key attribute until the last key attribute is compared with L-ATTR or until L-ATTR is equal to one of key attributes.

If L-ATTR is not equal to any key attributes

Compare L-ATTR with a set of non key attributes until L-ATTR is equal to one of non key attributes or until the last non key attribute is compared with L-ATTR.

If L-ATTR is not equal to any non key attributes put L-ATTR in the error attribute type 1.

Get each right attribute (R-ATTR) to compare with each key and non key attribute in the same way.

If R-ATTR is not equal to any keys or non keys put R-ATTR in the error attribute type 2.

Until the last dependency has been checked.

End while.

If the error attributes are found

Print error attributes into ERROR.FLE
Stop the process.
Otherwise go to the next step.

3. Construct new relations from 1-1 dependencies.
Search for 1-1 dependencies from the dependencies file
put them to 1-1 array.
Construct new relations from 1-1 array.
   the left attributes become primary key attributes
   the right attributes become non key attributes.
Insert new relations to the relations file.

4. Normalize into second normal forms.
The purpose of second normal forms is to remove partially
dependencies from a relation. Only the relations whose set
of primary keys contains more than one attributes are
possible to have partially dependencies.
While not end relations
   Read a relation into the relation array.
   Count number of key attributes.
If number of key attributes is greater than 1
   Search for the set of dependencies for this relation
If it is found
   Read the set of dependencies into the
   function array.
   Initialize dependency index to 0.
Repeat

Increase a dependency index by 1.

Compare a set of left attributes with key attributes.

If all left attributes are equal to a part of the key attributes

(* Decompose the relation *)

Construct a new relation from this dependency
the left attributes become key attributes
the right attributes become non keys.

Insert this new relation to the relation file.

Determine dependencies for a new relation.

Moving spaces to non key attributes which are equal to those right attributes.

Until the last dependency is checked.

End while.

5. Normalize into third normal form and fourth normal form.

Repeat

Call the subprogram "COMBIN" to combine relations with identical keys. (The detail of this subprogram is explained in the appendix E.)

While not end relations

Read a relation into a relation array.

Initialize a dependency index to 0.
Repeat

Increase a dependency index by 1.

Check primary keys with 1-1 array.

If the attributes in a set of primary keys
are equal to a set of right attributes in
1-1 array

Replace those attributes with left attributes
from 1-1 array.

Until all 1-1 dependencies have been checked.

End while.

While not end relations

Read a relation into the relation array.

Search for a set of dependencies corresponding to
this relation.

If it is found

Read a set of dependencies to function array.

Initialize a dependency index to 0.

Repeat

Increase a dependency index by 1.

If dependency type = '------>'

Check left and right attributes with
non key attributes

If they are equal

(* Decompose the relation *)

Construct a new relation from
this dependency.
Move space to non key attributes.

If dependent type = "->->"

Construct a new relation from this dependency.

Move spaces to the attributes which are equal to right attributes.

Until all dependencies have been checked.

Until no relations are decomposed.

6. Call subprogram "COMPRESS".
The purpose of this subprogram is to compress the relations file and dependencies file. A lot of unused records are in both files when some relations were deleted and some dependencies were used to decompose those relations. The subprogram copies only the records which do not have the delete tags in the front fields. The new relations file and new dependencies file are produced. The detail of this subprogram is explained in the appendix F.

7. Checked non key attributes with 1-1 array.
The purpose of this step is to look for non key attributes in a relation that can be replaced by their primary attributes. While not end relations

Read a relation into a relation array.

Initialize dependency index to 0.
Repeat

Increase dependency index by 1.

Check a set of right attributes in 1-1 array (FD-RIGHT) with non key attributes.

If FD-RIGHT is equal to non key attributes

Compare key attributes with a set of left attributes (FD-LEFT)

If a part of keys is equal to FD-LEFT

Replace non key attributes which are equal to FD-RIGHT with space.

If the whole keys are not equal to FD-LEFT

If a set of left attributes (FD-LEFT) does not appear in non key attributes

Replace non key attributes which are equal to FD-RIGHT with FD-LEFT attributes

Else Replace them with spaces.

Else next sentence.

Until the last 1-1 dependency has been checked with this relation.

If any attributes in this relation are changed

Write the new relation from the relation array.

Delete the old relation.

End while.

8. Display an optimal relational data model.

A relation name, key attributes and non key attributes are displayed respectively. The user is allowed to change a
relation name.

9. Allow to define more dependencies for each relation.

If a designer wants to modify a schema

While not end relations

Display a relation name, key attribute and non key attributes.

Repeat

If the designer wants to define more dependencies

Enter left attributes

Enter right attributes

Enter dependency type

Write this dependency to dependencies file.

Until the designer does not want to define any more.

End while.

10. Renormalize all relations.

If the designer has entered more dependencies from step 9

Go back to step 4

Else Go to next step.

11. Print the output to a print file.

Enter output file name.

Print the optimal relations.

Stop the execution.
Modify Old Database Schema

1. Get the value of id of the names file for the desired database.

Repeat
Accept a file name.
Check the existence of the file name by calling "MACRO" subprogram named "EXIST".
If the file name does not exist in a user's area
Display error messages.
Until the file name is correct in a user's area.
Open I-O name file.
Read name-file to get value of id of relation file.
Read name-file to get value of id of dependency file.

2. Accept the user's choice to modify the schema.
Accept A-CODE.
Check A-CODE until A-CODE = "1" or A-CODE = "2".
If A-CODE = "1" Modify by off-line
Else Modify by on-line.

Modify by Off-Line

This routine generates sequential files from the most recent random relation file and random dependency file of the desired database schema. The names of new sequential files are generated by changing the extension of those files to be "SEQ".
Open I-O relation file and dependency file.
Generate the value of id of output files.
Open output files: R-MODIFY, F-MODIFY.
Initialize R-KEY to 1.
Repeat
  Increase R-KEY by 1.
  Read a record from relation file.
  Write R-MODIFY record from relation file record.
Until the end of relation file is found.
Initialize F-KEY to 1.
Repeat
  Increase F-KEY by 1.
  Read a record from dependency file.
  Write F-MODIFY record from dependency file record.
Until the end of dependency file is found.
Display the new file names, some necessary messages.
Stop the execution.

Modify by On-Line

This routine allows a user define more dependencies for each relation directly to the system. The algorithm is same as described in step 9 through step 11 from the process of new database schema.
CONCLUSIONS

At the first phase of system design a user enters each relation along with a set of dependencies. After all relations have been entered, the programs normalize them into third normal form. If errors occur during the execution, the designer must start the process over. The system lets a user prepare the input files. The information of normalization is stored to the files. The input files are complete representation of all information for a given database. If some required symbols are missing, the process might get in an infinite loop. For example, if the statement "$ END RELATION" is missed, the execution will get into an infinite loop with the message "READ FUNCT INVALID". It means that the program cannot read a record from the dependencies file.

This thesis shows an attempt to implement the normalization procedure. The redundancies are eliminated while the schema can represent all the situations required in the database. The disadvantage is that there is a time delay when the program checks the attributes in each relation for a possible replacement with their primary attributes. This system is an efficient tool for adding the designer in the complex task of database design.
APPENDIX A

PROGRAM NAME : MAIN.CBL

This program is the main part of the project. It generates optimal relational data model for the given database. The program takes care of all steps of normalization. It also provides several options to serve the need of a user. The program consists of 4 subprograms as follow:

1. "SRNDOM" is called to convert the original input files into random files.

2. "SERROR" is called to find out some error attributes in the dependency file. Any attributes which are not matched to any attribute names in the relations file are considered to be errors.

3. "COMBIN" is called to combine relations with identical primary key attributes.

4. "CPRESS" is called to copy only the used records from the current relations file and dependency file into a new relations file and a new dependency file.

The output optimal relational data model is in the fourth normal form.
THE EXPLANATION OF ROUTINES

ROUTINE: ENTER-OPTION

A user is asked to enter a code to determine which database he wants to process. The code "1" means to process new database. The code "2" means to process an old database.

"A-CODE" is a variable used to accept the value of the code. The routine is performed until "A-CODE" value is either "1" or "2" otherwise the error message is shown and a user is asked to enter a new value.

ROUTINE: PROCESS-NEW-DATA

1. A user enters file names which he has created by using SOS.
   1. Relation file
   2. Function file
   3. Name file

The program performs the segment CHECK-FILE-NAME which does the following:

For each accepted a file name, this routine modifies the file name, if needed to generate a correct format and then call MACRO subprogram named "EXIST" to check for the existence of that file. The passing arguments are a file name, programmer name, project number, "ERRCODE". If
"ERRCODE" is equal to "0", it means that a file name is in a user area otherwise a user has entered a wrong name and he is asked to enter a new file name.

2. The subprogram "SRNDOM" generates random files from the original file created by a user.

3. The subprogram "SERROR" checks errors in a function file. If the variable "ERR-MESSAGE" is "true", the error messages are displayed on the screen and the program stops the process.

4. The subroutine BEGIN-MAIN does the following:
This routine performs the normalization from the first normal form through the fourth normal form. A user is allowed to define more dependencies for each relation at the end of normalization. The steps of the routine are described as following:

4.1 The subroutine GET-FILE-NAME does the following:

It gets the values of id for a relation file and a function file from the name file.
Read the first record of name file
Assign the value of "ID-NAME" to relation file name
Read the second record of name file
Assign the value of "ID-NAME" to function file name.
4.2 The subroutine READ-FIRST-RECORD does the following:

Reads the first record of the RELATION-FILE, stores the number of relations in to TOTAL-REL, stores the value of the last record in NEXT-R.

Reads the first record of FUNCTION-FILE, stores the value of last record in NEXT-R.

4.3 The subroutine GET-ONE-FD does the following:

The routine constructs one to one dependencies table from the FUNCTION-FILE.

It reads FUNCTION-FILE and checks each relation for the dependency type. If the dependency type is "--------->", copy the set of left attributes to "FD-LEFT" array. and copy the set of right attributes to "FD-RIGHT" array.

Delete the dependencies which have been copied to the table by putting "x" in the front field of each record.

4.4 The subroutine CHECK-FOR-2NF does the following:

a.) A relation is read in to a relation array which consists of a key attributes array, non key attributes array. The number of key attributes is counted and stored in "KEY-NUMBER".
b.) If the "KEY-NUMBER" is greater than 1 then a relation will be checked for second normal form otherwise it is not necessary to check. To check for second normal form, the dependencies associated with the relation in the relation array are read into a function array. A function array is declared in the form of an array of left attributes, an array of right attributes and the dependency type of a dependency. In case a user does not defined any associations, or functional dependencies for a relation, it means that a relation is already normalized.

c.) The subroutine PUT-IN-2NF does the following:

For each dependency, every left attribute is compared with a key attributes array. "EQUAL-ATTR" is a flag to indicate that a set of left attributes is equal to a subset of key attributes. "EQUAL-CTN" is increased by 1 for each comparison if a left attributes is equal to one of key attributes.

If "EQUAL-ATTR" = "true" and "EQUAL-CTN" is less than a number of key attributes, then the relation is decomposed.

A new relation resulted from the old relation based on the dependency which is a part of the key. The left attributes of the dependency become the keys and the right attributes become the non key attributes. This new relation
is written to the relation file. The attributes which are equal to a set of right attributes are removed from the old relation array by moving spaces to attribute names. At the end of executing CHECK-FOR-2NF all resulting relations are in the second normal form.

d.) The subroutine DETERMINE-NEW-DEPEND. This routine is a part of CHECK-FOR-2NF.

It determines dependencies for a newly created relation. It looks at each dependency in the table to compare a set of left attributes with a set of right attributes of a particular dependency (which formed a new relation). If they are equal, remove that dependency from the old functional dependency table and write it in the function file for a new relation.

Increase the index pointed to the next dependency. Repeat to start from step c.) again. ROUTINE PUT-IN-2NF is performed until the last dependency is processed.

e.) The subroutine CHECK-NEW-RELATION does the following:

After new relations are produced, the old relation will be deleted by putting 'Z' in the front field of a relation name. An old relation from a relation array whose some non key attributes are removed from the decompositions is written
to the relation file.

4.5 The subroutine SPLIT-ONE-ONE does the following:

This subroutine is performed in case there are one to one dependencies defined in a database. New relations are written from one to one table into relation file. The attributes stored in FD-LEFT(I) become key attributes and the attributes stored in FD-RIGHT(I) become non key attributes for a new relation. "I" is a index pointed to each dependency.

4.6 The subroutine PART-3NF does the following: The routine controls the loop of checking for third normal form. It calls "COMBINE" subprogram which combines relations with same keys. The subroutine "CHECK-FD-1-1" is performed to check for replacing key attributes with primary attributes from one to one table. It then performs CHECK-FOR-3NF.

The subroutine CHECK-FOR-3NF is a part of PART-3NF. It does the following:

a.) A relation is read into a relation array.

b.) Associated functional dependencies are read into a function array.
The subroutine PUT-IN-3NF does the following: The routine goes through each dependency to check the left attributes and the right attributes with a set of non-key attributes in a relation array. If both left attributes and right attributes are subsets of non-key attributes and the dependency type is equal to "----->" or "-->->" a new relation is produced from this dependency by the left attributes becoming keys of a relation the right attributes becoming non-key attributes.

The dependencies are determined for the new relation by DETERMINE-NEW-DEPEND routine as described in the previous routine (see 4.4 part d.).

d.) The subroutine CHECK-NEW-RELATION is performed. The detail of this routine is in 4.4 part e.

Repeat CHECK-FOR-3NF routine again for a next relation until the end of the relation file is found.

The subroutine CHECK-FD-1-1 goes through all relations if there was any decomposition when CHECK-FOR-3NF was last executed.

a.) Key attributes of each relation are compared with the right attributes stored in FD-RIGHT(I) from one to one table. If some parts of key attributes are equal to the set of right attributes in the table those parts of key...
attributes will be replaced by spaces and the set of left attributes in FD-LEFT(I) are inserted one by one to a next position of the key attributes array.

b.) Increase an index "I" to point to a next one to one dependency in the table and repeat step a. again until the last one to one dependency is compared.

c.) Read a next relation to a relation array repeat steps a, b again until the end of file is found.

4.7 Call subprogram "COMPRESS" to generate a new relation file and a new function file. A new relation file contains all relations which are normalized in third normal form. A new function file contains functional dependencies for those relations.

4.8 The subroutine CHECK-NON-KEY-1-1 replaces a non key attribute by its equivalent primary attributes. The following steps are executed.

a.) A relation is read into a relation array

b.) It compares a set of right attributes stored in FD-RIGHT of one to one table with each non key attribute in a relation array. If a set of right attributes in FD-RIGHT is a subset of non-key attributes being compared, one of these cases will happen.
case 1: The set of non key attributes which is equal to FD-RIGHT is not replaced if the corresponding FD-LEFT is the key.

case 2: If a set of non key attributes is one to one to another set of non key attributes, the non primary set is removed.

Case 3: The set of non key attributes which is equal to FD-RIGHT(I) is replaced by the set of left attributes in FD-LEFT if case 1 or case 2 does not happen.

c.) Increase index "I" by 1 to point to a next one to one dependency in the table, the go to step b. again.

d.) Repeat from step a. again to get a next relation until the end of file is found.

4.9 The subroutine DISPLAY-ANSWER does the following:

Display the normalized relations on the screen. a relation name, key attributes, and non key attributes are shown respectively. A user is allowed to change a relation name for each relation.

4.10 The subroutine ADJUST-NORMAL-FORM does the following:
Allow the user to define more dependencies for each relation. If a user desires to define more, a user is to enter left attributes, right attributes and dependency type. The program will prompt the user. The control goes back to start from step 4.4 through step 4.9 again. The loop is repeated until a user does not want to define any more dependencies.

5. The subroutine PRINT-OUTPUT does the following:

The user enters the report file name. The normalized relational data model is printed to the report file If there are one to one dependencies in one to one table, they are printed at the end of the output as a remark.

ROUTINE: PROCESS-OLD-DATA

A user enters a name of NAME-FILE for the old database. CHECK-FILE-NAME routine is performed to check the existence of the file name as described at the beginning.

The routine provides two options to modify the schema. the first option is to modify by off-line. The second option is to modify by on-line. The user enters the option code. A-CODE is a variable to accept the value of the code. The routine ENTER-OPTION is performed as described in PROCESS-NEW-DATABASE subroutine.
The program performs the segment MODIFY-OFF-LINE if the code = "1". It does the following:

Perform GET-FILE-NAME

Open I-O relation file and function file.

Perform R-MODIFY routine which generate a new sequential relation file from the random input relation file.

Perform F-MODIFY routine which generate a new sequential function file from the random input function file.

Display new file names.

Stop the execution.

The program performs the segment MODIFY-ON-LINE if the code = "2". The routine ADJUST-NORMAL-FORM is performed as described in the previous part.
APPENDIX B
SUBPROGRAM NAME : CONVRT.CBL

I. PURPOSE

The function of a subprogram is to convert the original sequential files which a user has created by SOS to random files. The three input files are as following.

1. Relations file
2. Function file
3. Names file

All files must be in random accesses because programs will be able to insert new records or update records in the files. The subprogram was written in COBOL language.

II. PASSING ARGUMENTS

NAME-1 is stored a value of id of relation file.
NAME-2 is stored a value of id of function file.
NAME-3 is stored a value of id of name file.

III. ALGORITHMS

1. Get value of ids of three input files from the passing arguments.

\[
\begin{align*}
\text{INPUT-1} & = \text{NAME-1} \\
\text{INPUT-2} & = \text{NAME-2}
\end{align*}
\]
INPUT-3 = NAME-3

Open the three input files; relation file, function file, names file;  
Generate a value of id of output name file by changing the extension of INPUT-3 to be "001"; 
Open I-0 names file;

2. Perform segment WRITE-FILE-NAME which does the following: 
   Initialize NAME-FILE-KEY to 0; 
   While not the end of names file 
      read a record from input name file;  
      NAME-FILE-KEY = NAME-FILE-KEY + 1; 
      write an output record from the input record;  
   End while.

3. Perform segment GET-FILE-NAME which does the following: Get the value of id of output relation file from the first record of names file and get the value of id of output function file from the second record of names file. 
   NAME-FILE-KEY = 1;  
   read a record from names file;  
   INPUT-1 = ID-NAME; 
   NAME-FILE-KEY = 2; 
   read a record from names file;  
   INPUT-2 = ID-NAME;
open I-O RELATION-FILE, FUNCTION-FILE;

4. Perform segment READ-W-FILE-1 which does the following: read a record from the input relation file and write it to the output relation file one by one and also count the number of relations in the file.

   Initialize R-KEY to 0;
   Initialize TOTAL-REL to 0;

   Repeat
     read a record from an input relation file;
     write a record to output relation file from an input record;
     If CHAR1 = '/' then
       TOTAL-REL = TOTAL-REL + 1;
       R-KEY = R-KEY + 1;
   Until the end of file is found;

   (* Write the first record *)
   LAST-R-RECORD = R-KEY - 2;
   NUMBER-REL = TOTAL-REL;
   R-KEY = 1;
   write the first record of an output relation file;

5. Perform segment READ-W-FILE-2 which does the following: read a record from the input function file and write it to output function file one by one.
Initialize F-KEY to 2;
Repeat
  read a record from an input function file;
  write a record from an input record into an output function file;
  F-KEY = F-KEY + 1;
Until the end of file is found;

6. Move the value of id of output NAME-FILE to the third argument.

7. Close files.

8. Return to the main program.

IV. ENTRY POINTS

There is one entry point at the beginning of the procedure division.

V. EXIT POINTS

There is one exit point at the end of the main routine.
APPENDIX C
SUBPROGRAM : ERR.CBL

I. PURPOSE

The subprogram was written to find basic errors which often come from typing errors. It assumes that the attribute names are unique, and the set of functional dependencies forms a cover.

1. The left attributes are composed of key attributes and non key attributes. A user should review a dependency again because sometimes the error does not come from typing error but a user defined a redundant dependency.

2. Some left attributes in a dependency do not exist in either a set of key attributes or a set of non key attributes. A user probably typed wrong attribute names.

3. Some right attributes in a dependency do not exist in either a set of key attributes or a set of non key attributes. A user probably typed wrong attribute names.

II. PASSING ARGUMENTS

NAME-FILE-VALUE-ID is used to store a
value of id of name file.

ERR-MESSAGE is a flag to indicate that there are
errors found in a function file.

III. ALGORITHMS

1. GET-FILE-NAME. It gets the value of id of the
relation file and the value of id of the function file from
the name file.

   move NAME-FILE-VALUE-ID to a value of id
   of NAME-FILE;
   open I-0 NAME-FILE;
   NAME-FILE-KEY = 1;
   read the first record of the name file;
   FILE-NAME-1 = ID-NAME;
   NAME-FILE-KEY = 2;
   read the second record of the name file;
   FILE-NAME-2 = ID-NAME;
   close name file;
   open I-0 relation file, function file.

2. Initialize ERR-MESSAGE to space;

3. ERROR-CHECKING

Repeat
read a relation file into a relation array
searching for functional dependencies defined
for a relation;
If they are found then
read all dependencies for that relation
into a function table;
I = 1;
(* I is an index pointed to each dependency *)
repeat
initialize error index E1, E2, E3, E4;
KEY-CTN = 1;
(* points to first key attributes in the relation array *)
LEFT-CTN = 1;
(* points to first left attribute currently being checked *)
CHECK-LEFT-ATTR;
ATTR-CTN = 1;
(* points to first non key attribute currently being checked *)
RIGHT-CTN = 1;
(* points to first right attribute currently being checked *)
CHECK-RIGHT-ATTR;
(* check error cases *)
case :
(E1 > 0) and (E2 > 0) : error case 1
E3 > 0 : error case 2
E5 > 0 : error case 3

If errors occur then

ERR-MESSAGE = 'true'

write error messages, attribute names;

(* increase I pointed to next dependency *)

I = I + 1;

until LEFT-ATTR(I, 1) = 'ZZ';

R-KEY = R-KEY + 1;

Until the end of relation file is found.


5. Return to the main program.

THE ALGORITHM OF CHECK-LEFT-ATTR

Repeat

repeat

(* compare a left attribute in FUNCTION file
with each key attribute in relation file *)

If LEFT-ATTR(I, LEFT-CTN) = KEY-ATTR(KEY-CTN)

EQUAL is 'true'

E1 = E1 + 1

store a left attribute to ER1 array

else

KEY-CTN = KEY-CTN + 1
until EQUAL or KEY-CTN > KEY-NUMBER;

If not equal then
(* compare it with a non key attribute *)
 repeat
   if LEFT-ATTR(I, LEFT-CTN) = ATTR-NAME(ATTR-CTN)
      E2 = E2 + 1
      store a left attribute to ER2 array
   else ATTR-CTN = attr-ctn + 1;
   until EQUAL or ATTR-NAME(ATTR-CTN) = 'ZZ';
If not EQUAL then
   it is error case 2
   E3 = E3 + 1
   store a left attribute in ER3 array;
(* increase an index pointed to a next left attribute *)
   LEFT-CTN = LEFT-CTN + 1;
 Until LEFT-ATTR(I, LEFT-CTN) = 'ZZ'.

THE ALGORITHM OF CHECK-RIGHT-ATTR

Repeat
    repeat
(* compare a right attribute with a key attribute *)
 If RIGHT-ATTR(I, RIGHT-CTN) = KEY-ATTR(KEY-CTN)
    E4 = E4 + 1
    store a right attribute in ER4 array

EQUAL is 'true'
else
    KEY-CTN = KEY-CTN + 1
until KEY-CTN > KEY-NUMBER;
If not EQUAL then
(* compare a right attribute with a non key *)
repeat
    if RIGHT-ATTR(I, RIGHT-CTN) = ATTR-NAME(ATTR-CTN)
        EQUAL is 'true'
    else ATTR-CTN = attr-ctn + 1
until EQUAL or ATTR-NAME(ATTR-CTN) = 'ZZ';
if not EQUAL then
    it is error case 3
    ER5 = ER5 + 1
    store a right attribute in ER5 array;
(* increase an index pointed to a next right attribute *)
    RIGHT-CTN = RIGHT-CTN + 1
Until RIGHT-ATTR(I, RIGHT-CTN) = 'ZZ'.

IV. ENTRY POINTS

There is one entry point at the beginning of the procedure division.

V. EXIT POINTS

There is one exit point at the end of the main routine.
APPENDIX D

SUBPROGRAM : COMB.CBL

I. PURPOSE

To eliminate redundancy, this subprogram is called to combine relations which have identical key attributes to one relation. Their functional dependencies are put together. The subprogram is called whenever there are decompositions of relations or when key attributes of any relation are replaced by primary attributes.

II. PASSING ARGUMENTS

NAME-FILE-VALUE-ID is used to store a value of id of name file.

COMB-VALUE is a flag to indicate that relations are combined together.

III. ALGORITHMS

1. GET-FILE-NAME This routine performs a function as described in the subprogram ERR.CBL.

2. Initialize COMB-VALUE, END-COMB, END-POINT to space.
END-COMB is a flag set to stop combine loop.

END-POINT is used to store the record number which is the first record of the first combined relation (To prevent unnecessary checks of relations to be combined).

3. COMBINE-LOOP

Repeat

repeat

read a relation file and get a relation into a relation array
until the end of relation is found or the end of file or END-COMB = 'true';

move R-KEY to TEMP-KEY;

If not end of file and not END-COMB find functional dependencies for a relation
repeat

read until the next relation is found
compare key attributes in the array with a key attribute in the record
if all key attributes are equal COMB = 'true'
repeat

read the next record

move name to attribute array
until the end of relation;
delete this relation;
determine new dependencies by
moving each dependency to function array

R-KEY = R-KEY + 1
until the end of file or R-KEY = END-POINT
If COMB = 'true'

write the new relation from the array

delete the old relation;
R-KEY = TEMP-KEY;
Until the end of file or END-COMB = 'true'.

END-POINT is set when the first new relation is produced. It stores the value of the first record number of the first new combined relation.

4. WRITE-LAST-RECORD This routine updates the number of relations and the last record number because they are changed when new relations are added to the file and some relations are logically deleted. The routine also updates a last record number of a function file. This is executed after "COMBINE-loop is finished. (During the execution of COMBINE-loop the number of relations and the position of last record are constantly updated.)

LAST-R-RECORD = NEXT-R;
NUMBER-RELS = TOTAL-REL
R-KEY = 1;
write the first record of a relation file;
last-f-record = NEXT-F;
F-KEY = 1;
write the first record of a function file;
(which contains this general information)

5. Close files.

6. Return to the main program

IV. ENTRY POINTS

There is one entry point at the beginning of the procedure division.

V. EXIT POINTS

There is one exit point at the end of the main routine.
I. PURPOSE

The CPRESS.CBL subprogram is called to produce a new relation file, a new function file and to update records in a name file. After the main program normalized relations to third normal form, a relation file and a function file contain some useless records which have '%' in the front field of relation names because some relations are decomposed, others are combined and new relations are produced. The files become large and might exceed the file limits so it is necessary to generate a new relation file and a new function file which do not contain useless records. The output files have the same format of file descriptions as input files do.

II. PASSING ARGUMENTS

NAME-FILE-VALUE-ID is used to store a value of id of name file.

III. ALGORITHMS
1. Initialize relative keys.

R-KEY = 0
(* a relative key of relation file *)

F-KEY = 0
(* a relative key of function file *)

NAME-FILE-KEY = 0
(* a relative key of name file *)

RNEW-KEY = 0
(* a relative key of a new relation file *)

FNEW-KEY = 0
(* a relative key of a new function file *)

2. GET-FILE-NAME. This routine performs a function as described in the subprogram ERR.CBL

3. Get file names for output files.

NAME-FILE-KEY = 3;
read the third record of the name file;
move ID-NAME to value of id of a new relation file;

NAME-FILE-KEY = 4;
read the fourth record of the name file;
move ID-NAME to value of id of a new function file;
open I-O new relation file, new function.

4. COMPRESS-REL
Repeat

R-KEY = R-KEY + 1
read a relation file
If the first field of a record = '/'
(* find the beginning of a relation *)
TOTAL-REL = TOTAL-REL + 1
RNEW-KEY = RNEW-KEY + 1

copy a record to a new relation file
repeat
  RNEW-KEY = RNEW-KEY + 1
  R-KEY = R-KEY + 1
  read next record and copy it to a new file
  until the end of relation is found;
Until the end of file is found;

(* write the number of relations,
the number of the last record *)
NUMBER-RELS = TOTAL-REL
LAST-R-RECORD = RNEW-KEY
RNEW-KEY = 1
write the first record of a new relation file.

5. COMPRESS-FUNC
Repeat
  F-KEY = F-KEY + 1
  read a function file
  If the first field of a record = '/'
  FNEW-KEY = FNEW-KEY + 1
copy a record to a new function file
repeat
  F-KEY = F-KEY + 1
read next record of function file
  FNEW-KEY = FNEW-KEY + 1
  copy a next record to a new function file
to until the end of relation is found;

  Until the end of file is found.

  (* write the number of records *)
  LAST-F-RECORD = FNEW-KEY
  FNEW-KEY = 1

  write the first record of a new function file.


7. WRITE-FILE-NAME
   NAME-FILE-KEY = 1
   ID-NAME = value of id of a new relation file
   write the first record of a name file
   NAME-FILE-KEY = 2
   ID-NAME = value of id of a new function file
   write the second record of a name file
   NAME-FILE-KEY = 3
   assign a new name for the third record by changing the last character.
write the third record of a name file
assign a new name for the fourth record by
changing the last character.
write the fourth record of a name file.

8. Close a name file.
9. Return to the main program.

IV. ENTRY POINTS

There is one entry point at the beginning of the
procedure division.

V. EXIT POINTS

There is one exit point at the end of the main routine.
APPENDIX F

THE EXAMPLE OF THE COMPLETE RUNNING PROCEDURE

.RUN DESIGN

A DESIGN OF OPTIMAL RELATION DATA MODEL

THE PROGRAM PROVIDES TWO OPTIONS FOR THE USER.

1. PROCESS NEW DATA BASE
2. MODIFY YOUR OLD DATA BASE

PLEASE ENTER THE CODE YOU DESIRED ("1" OR "2"): 1

PROCESS YOUR NEW DATA BASE

ENTER THE FILE NAMES YOU HAVE CREATED BY SOS

1. RELATION FILE = RHOSP.FLE
2. FUNCTION FILE = FHOSP.FLE
3. NAMES FILE = N.FLE

** THIS FILE IS NOT IN YOUR AREA

ENTER NEW NAME = NHOSP.FLE

STEP I. CONVERT THE ORIGINALE FILES TO RANDOM FILES

** END OF STEP I.

STEP II. CHECK ERROR ATTRIBUTES IN DEPENDENCIES FILE

ENTER ERROR OUTPUT FILE NAME = ERRORS.FLE

85
** END OF STEP II.**

** NO ERRORS FOUND AT THIS TIME**

** STEP III. NORMALIZE RELATIONAL DATA MODEL **

** NORMALIZED IN SECOND NORMAL FORM**

!! PLEASE WAIT A WHILE. !!
!! THE PROGRAM IS WORKING FOR THIRD NORMAL FORM. !!
!! AND FOURTH NORMAL FORM !!

** NORMALIZED IN FOURTH NORMAL FORM**

** OPTIMAL RELATIONAL DATA MODEL **

** RELATION NAME = 1ONE-TO-ONE-FD **

KEY ATTRIBUTE : P-ID
NON KEY ATTRIBUTE : SS-NO
P-ADDR
P-NAME

** DO YOU WANT TO CHANGE RELATION NAME ? (Y/N) : Y **

ENTER NEW RELATION NAME : PATIENT-FILE

** RELATION NAME = 1PATIENT-OPERATION **

KEY ATTRIBUTE : DOCTOR
NON KEY ATTRIBUTE : SPECIALIZE-IN

** DO YOU WANT TO CHANGE RELATION NAME ? (Y/N) : Y **
ENTER NEW RELATION NAME : DOCTOR-FILE

RELATION NAME = 2PATIENT-OPERATION
KEY ATTRIBUTE : P-ID
   OP-DATE
NON KEY ATTRIBUTE : PART-OPERATE
   DOCTOR
   LICENSE-NO
   RESULT
   SIDE-EFFECT

** DO YOU WANT TO CHANGE RELATION NAME ? (Y/N) : Y

ENTER NEW RELATION NAME : PATIENT-OPERATION

RELATION NAME = 1PATIENT-ADMIT
KEY ATTRIBUTE : P-ID
   ADMIT-DATE
NON KEY ATTRIBUTE : SECTION-ADMIT
   SYMTOMS
   TREATMENT
   DOCTOR

** DO YOU WANT TO CHANGE RELATION NAME ? (Y/N) : Y

ENTER NEW RELATION NAME : PATIENT-ADMITANCE

** DO YOU WANT TO LOOK OVER AGAIN ? (Y/N) : N

** DO YOU WANT TO DEFINE MORE DEPENDENCIES ? (Y/N) : N
ENTER THE REPORT FILE NAME = HOSP.RPT

** END OF THE NORMALIZATION

** THE NAME-FILE FOR THIS DATABASE IS NHOSP.001

** REMEMBER THE NAME-FILE FOR A NEXT PROCESS

** END OF EXECUTION **

******************************

EXIT
## OPTIMAL RELATIONAL DATA MODEL

### RELATION NAME = PATIENT-FILE

<table>
<thead>
<tr>
<th>P-ID</th>
<th>SS-NO</th>
<th>P-ADDR</th>
<th>P-NAME</th>
</tr>
</thead>
</table>

### RELATION NAME = DOCTOR-FILE

<table>
<thead>
<tr>
<th>DOCTOR</th>
<th>SPECIALIZE-IN</th>
</tr>
</thead>
</table>

### RELATION NAME = PATIENT-OPERATION

<table>
<thead>
<tr>
<th>P-ID</th>
<th>OP-DATE</th>
<th>PART-OPERATE</th>
<th>DOCTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>LICENSE-NO</td>
<td>RESULT</td>
<td>SIDE-EFFECT</td>
<td></td>
</tr>
</tbody>
</table>

### RELATION NAME = PATIENT-ADMIT

<table>
<thead>
<tr>
<th>P-ID</th>
<th>ADMIT-DATE</th>
<th>SECTION-ADMIT</th>
<th>SYMPTOMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>TREATMENT</td>
<td>DOCTOR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
REMARKS:

THE LIST OF ONE TO ONE DEPENDENCIES

1. PRIMARY ATTRIBUTES:
   P-ID
   CANDIDATE ATTRIBUTES:
   SS-NO

2. PRIMARY ATTRIBUTES:
   P-ID
   CANDIDATE ATTRIBUTES:
   P-ADDR
   P-NAME

** END OF OUTPUT **

Figure 1
The original input relations file: RHOSP.FLE

/ PATIENT-ADMIT
# P-ID
# ADMIT-DATE
* P-NAME
* P-ADDR
* SS-N0
* SECTION-ADMIT
* SYMTPMS
* TREATMENT
* DOCTOR
$ END RELATION
/ PATIENT-OPERATION
# P-ID
# OP-DATE
* PART-OPERATE
* DOCTOR
* LICENSE-NO
* SPECIALIZE-IN
* RESULT
* SIDE-EFFECT
$ END RELATION
$ EOF
The original input dependencies file: FHOSP.FLE

/ PATIENT-ADMIT
L* P-ID
;
R* P-NAME
R* P-ADDR
;
R* SS-NO
<----->
$ END RELATION
/ PATIENT-OPERATION
L* DOCTOR
R* SPECIALIZE-IN
----->
$ END RELATION
$ EOF
The original input names file: NHOSP.FLE

RHOSP RND
FHOSP RND
RHOSP OUT
FHOSP OUT
The example of the modification by on-line.

.RUN DESIGN

A DESIGN OF OPTIMAL RELATION DATA MODEL
******************************************************************************

THE PROGRAM PROVIDES TWO OPTIONS FOR THE USER.

1. PROCESS NEW DATA BASE
2. MODIFY YOUR OLD DATA BASE

PLEASE ENTER THE CODE YOU DESIRED ("1" OR "2") : 2

MODIFY YOUR OLD DATA BASE
==========================

ENTER "NAMES-FILE" : NHOSP.001

THIS MODULE HAS TWO OPTIONS AS FOLLOW:

1. MODIFY THE DATABASE SCHEMA BY OFF-LINE
2. MODIFY THE DATABASE SCHEMA BY ON-LINE

PLEASE ENTER THE CODE YOU DESIRED ("1" OR "2") : 2

** DO YOU WANT TO DEFINE MORE DEPENDENCIES? (Y/N) : Y

STEP IV. A USER DEFINES MORE DEPENDENCIES
=================================================

RELATION NAME = PATIENT-FILE

KEY ATTRIBUTE : P-ID

NON KEY ATTRIBUTE : SS-NO
P-ADDR
P-NAME
** ANY DEPENDENCIES FOR THIS RELATION ? (Y/N) : N

RELATION NAME = DOCTOR-FILE

KEY ATTRIBUTE : DOCTOR

NON KEY ATTRIBUTE : SPECIALIZE-IN

** ANY DEPENDENCIES FOR THIS RELATION ? (Y/N) : N

RELATION NAME = PATIENT-OPERATION

KEY ATTRIBUTE : P-ID
       OP-DATE

NON KEY ATTRIBUTE : PART-OPERATE
                   DOCTOR
                   LICENSE-NO
                   RESULT
                   SIDE-EFFECT

** ANY DEPENDENCIES FOR THIS RELATION ? (Y/N) : Y

ENTER LEFT ATTRIBUTES (TYPE "$" TO STOP)

: DOCTOR
 : $

ENTER RIGHT ATTRIBUTES (TYPE "$" TO STOP)

: LICENSE-NO
 : $

ENTER DEPENDENCY TYPE : <----->

** DO YOU WANT TO CANCEL? (Y/N) : N

** ANY MORE DEPENDENCIES ? (Y/N) : N

RELATION NAME = PATIENT-ADMITANCE
KEY ATTRIBUTE: P-ID
   ADMIT-DATE

NON KEY ATTRIBUTE: SECTION-ADMIT
   SYMTOMS
   TREATMENT
   DOCTOR

** ANY DEPENDENCIES FOR THIS RELATION? (Y/N): N

** NORMALIZED IN SECOND NORMAL FORM

** NORMALIZED IN FOURTH NORMAL FORM

** DO YOU WANT TO DEFINE MORE DEPENDENCIES? (Y/N): N

OPTIMAL RELATIONAL DATA MODEL

RELATION NAME = PATIENT-FILE

KEY ATTRIBUTE: P-ID

NON KEY ATTRIBUTE: SS-NO
   P-ADDR
   P-NAME

** DO YOU WANT TO CHANGE RELATION NAME? (Y/N): N

RELATION NAME = PATIENT-ADMITANCE

KEY ATTRIBUTE: P-ID
   ADMIT-DATE

NON KEY ATTRIBUTE: SECTION-ADMIT
   SYMTOMS
   TREATMENT

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
** DO YOU WANT TO CHANGE RELATION NAME ? (Y/N) : N

RELATION NAME = 1DOCTOR-FILE
KEY ATTRIBUTE : DOCTOR
NON KEY ATTRIBUTE : SPECIALIZE-IN LICENSE-NO

** DO YOU WANT TO CHANGE RELATION NAME ? (Y/N) : Y
ENTER NEW RELATION NAME : DOCTOR-FILE

RELATION NAME = 1PATIENT-OPERATION
KEY ATTRIBUTE : P-ID OP-DATE
NON KEY ATTRIBUTE : PART-OPERATE DOCTOR RESULT SIDE-EFFECT

** DO YOU WANT TO CHANGE RELATION NAME ? (Y/N) : Y
ENTER NEW RELATION NAME : PATIENT-OPERATION

** DO YOU WANT TO LOOK OVER AGAIN ? (Y/N) : N
ENTER THE REPORT FILE NAME = HOSP.RPT

** END OF THE NORMALIZATION
** THE NAME-FILE FOR THIS DATABASE IS NHOSP.001

** REMEMBER THE NAME-FILE FOR A NEXT PROCESS

** END OF EXECUTION **

*******************

EXIT
### OPTIMAL RELATIONAL DATA MODEL

<table>
<thead>
<tr>
<th>Relation Name</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PATIENT-FILE</td>
<td>P-ID, SS-NO, P-ADDR, P-NAME</td>
</tr>
<tr>
<td>PATIENT-ADMITANCE</td>
<td>P-ID, ADMIT-DATE, SECTION-ADMIT, SYMTOMS</td>
</tr>
<tr>
<td>DOCTOR-FILE</td>
<td>DOCTOR, SPECIALIZE-IN, LICENSE-NO</td>
</tr>
<tr>
<td>PATIENT-OPERATION</td>
<td>P-ID, OP-DATE, PART-OPERATE, DOCTOR</td>
</tr>
<tr>
<td></td>
<td>RESULT, SIDE-EFFECT</td>
</tr>
</tbody>
</table>
REMARKS:

THE LIST OF ONE TO ONE DEPENDENCIES

1. PRIMARY ATTRIBUTES:
   DOCTOR
   CANDIDATE ATTRIBUTES:
   LICENSE-NO

** END OF OUTPUT **
***************
The example of the modification by off-line.

.RUN DESIGN

A DESIGN OF OPTIMAL RELATION DATA MODEL

THE PROGRAM PROVIDES TWO OPTIONS FOR THE USER.

1. PROCESS NEW DATA BASE
2. MODIFY YOUR OLD DATA BASE

PLEASE ENTER THE CODE YOU DESIRED ("1" OR "2") : 2

MODIFY YOUR OLD DATA BASE

ENTER "NAMES-FILE" : NHOSP.001

THIS MODULE HAS TWO OPTIONS AS FOLLOW :

1. MODIFY THE DATABASE SCHEMA BY OFF-LINE
2. MODIFY THE DATABASE SCHEMA BY ON-LINE

PLEASE ENTER THE CODE YOU DESIRED ("1" OR "2") : 1

** TO MODIFY THE OLD DATABASE SCHEMA OFF-LINE
** THE FOLLOWING FILE NAMES CAN BE MODIFIED BY USING "SOS"
   THE RELATIONS FILE NAME IS RHOSP.SEQ
   THE FUNCTION FILE NAME IS FHOSP.SEQ

** THE PROGRAM WILL EXIT TO LET YOU MODIFY THE FILES.
** PLEASE REMEMBER,
   YOU ARE TO CHOICE THE OPTION "1" TO RUN
   THE MODIFIED INPUT FILES.

   YOU ALSO HAVE TO CREATE A NAME-FILE FOR
   YOUR NEXT EXECUTION.

   THE NAMES OF THOSE FILES CAN BE RENAMED BY YOU LATER.
** END OF EXECUTION **

***************

EXIT
The example of running procedure when there are errors attributes.

.RUN DESIGN

A DESIGN OF OPTIMAL RELATION DATA MODEL

THE PROGRAM PROVIDES TWO OPTIONS FOR THE USER.

1. PROCESS NEW DATA BASE
2. MODIFY YOUR OLD DATA BASE

PLEASE ENTER THE CODE YOU DESIRED ("1" OR "2") : 1

PROCESS YOUR NEW DATA BASE

ENTER THE FILE NAMES YOU HAVE CREATED BY SOS

1. RELATION FILE = RHOSP.FLE
2. FUNCTION FILE = FHOSP.FLE
3. NAMES FILE = NHOSP.FLE

STEP I. CONVERT THE ORIGINAL FILES TO RANDOM FILES

** END OF STEP I.

STEP II. CHECK ERROR ATTRIBUTES IN DEPENDENCIES FILE

ENTER ERROR OUTPUT FILE NAME = EHOSP.RPT

** END OF STEP II.
* THE ORIGINAL FILES SHOULD BE REVIEWED
** YOU MUST GO BACK TO START AT THE BEGINNING
** CANNOT CONTINUE THE PROCESS ...

** LOOK IN YOUR ERROR OUTPUT FILE

EXIT
The original relations file: RHOSP.FLE

/ PATIENT-ADMIT
  # P-ID
  $ ADMIT-DATE
  * P-NAME
  * P-ADDR
  * SS-NO
  * SECTION-ADMIT
  * SYMTPMS
  * TREATMENT
  * DOCTOR
  $ END RELATION
/ PATIENT-OPERATION
  # P-ID
  $ OP-DATE
  * PART-OPERATE
  * DOCTOR
  * LICENSE-NO
  * SPECIALIZE-IN
  * RESULT
  * SIDE-EFFECT
  $ END RELATION
  $ EOF
original dependencies file: FHOSP.FLE

/ PATIENT-ADMIT
L* P-ID
;
R* P-NAME
R* ADDR
;
R* SS-NO
<---->
$ END RELATION
/ PATIENT-OPERATION
L* DOCTOR
R* SPECIALIZE-IN
<---->
L* DOCTOR
;
R* LICENSE-NO
<---->
$ END RELATION
$ EOF
ERROR CHECKING FOR DEPENDENCY FILE
--------------------------------------------------

PATIENT-ADMIT
--------------------------------------------------

FD# 1 ** THE FOLLOWING RIGHT ATTRIBUTES DO NOT EXIST IN THIS RELATION **
ADDR

PATIENT-OPERATION
--------------------------------------------------

FD# 1 ** THE FOLLOWING LEFT ATTRIBUTES DO NOT EXIST IN THIS RELATION **
Dotor

FD# 2 ** THE FOLLOWING RIGHT ATTRIBUTES DO NOT EXIST IN THIS RELATION **
LICENSE-NO

** END OF OUTPUT **
********************************************

Figure 2
BIBLIOGRAPHY


108

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.