National Library of Canada
Canadian Theses Service
Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Services des thèses canadiennes

CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

NL 339 (r. 86/01)

Canada

AUTOMATED DESIGN OF BINARY RELATIONAL DATA BASES

by

Ramez Khatib

A thesis
presented to the University of Ottawa
in partial fulfillment of the
requirements for the degree of
MASTER OF SCIENCE
in
SYSTEMS SCIENCE

© Ramez Khatib, Ottawa, Canada, 1983.

CONTENTS

TABLE &

Chapter I

PRELIMINARIES

1.1 INTRODUCTION

In the last few years, extensive efforts have been invested in the field of data processing. The major problem faced by researchers has been: how to handle and manipulate efficiently large amounts of data. As relations between data become more and more complex, traditional file management systems become increasingly less efficient. To overcome this problem, many data base systems were designed. As stated in (Ma), a data base system may be defined as a collection of interrelated data stored without harmful or unnecessary redundancy to serve multiple applications; the data are stored so that they are independent of the programs

which use the data. Several data base management systems are now commercially available. The main functions of these systems is to define and manipulate the data base conveniently while striving to achieve the main objectives of data base technology: data independence, non-redundancy, and relatability. Unfortunatly, no precise measure for any of these objectives has been established, and any analysis with respect to one of them is mostly based on intuitive observations. Three approaches have been adopted to achieve these objectives. These are the network approach;the hierarchic approach; and the relational approach. The data bases designed using these approaches are called network, hierarchical, and relational, respectively.

Relational data bases are relatively easy to understand and to handle. The relational approach was first conceived and studied by Codd(Co1,Co2,Co3,Co4). Informally, a relational data base is a collection of relations. A relation is a two-dimensional table with n columns and a set of rows, each row being a n-tuple. All the values in a column are of the same type, and each column is called a domain. The type of the data item is called attribute. Each tuple of a rela-

tion must have a key by which the tuple can be uniquely identified and differentiated from other tuples of the same relation. Manipulations on relations can be performed by means of operators to extract or to join domains. For a more detailed descriptions of relational data bases see (Ca,Da,Ma,Ull).

Under the relational approach, the designers of a data base system identify and define a set of relations, called the basis, which is explicitly stored. Other relations can be dynamically formed from the basis by applying some operations to its elements. This structure makes the data base more flexible, but at the same time increase the complexity of the software needed to handle the data, such as the Data Definition Language and the Information Retrieval System. Since, as mentioned above, only the basis is actually stored, two questions arise here:

a. What is the minimum set of relations which when explicitly stored, allows all the remaining relations to be formed by performing some operations (such as join, project, restrict, etc.) on the elements of this set.

b. What is the set of relations which when explicitly stored, permit all the remaining relations to be formed with a minimum response time.

To our knowledge there is no systematic way to answer either of these questions for a general relational data base; the design of an efficient basis is only intuitive and is based on the observations of the designer. Some investigations in this area include (Lo) and Mitoma and Irani (Mi) who developed a methodology that facilitates the automatic design of Optimized DBTG schema structures for specified data management applications.

In this thesis we consider a particular class of relational data bases, the binary relational data bases. As for general relational data bases, a binary relational data base is a collection of binary relations. For this class of data bases, we show that using some theoretical results obtained in (Sa), the two questions posed above can be answered in a systematic way.

1.2 OUTLINE OF THE THESIS

This thesis focuses on the automated determination of the basis of a binary relational data base, and on the practical implications of the binary relational model.

In the next chapter, the binary relational model and some theoretical results obtained in (Sa) are presented. Relational graphs are used to build up the set of relations and to determine the set of constraints; this determination plays a critical role in the construction of efficient abstract structures.

In chapter three it is shown that the integrity problem is avoided by using the constraints; a procedure to build the set of constraints is given, and a technique to build the conceptual model is described. The theoretical results presented in chapter 2 are used to determine efficient abstract structures. All these concepts are illustrated by means of examples. Finally an overall methodology for the design of space and time efficient abstract structures is described.

Chapter four is a detailed example of the automated construction of efficient abstract structures. A software package, implementing the methodology described in the thesis, is applied to a conceptual model describing an 'academic data base'. As a result, a time/space efficient abstract structure implementing the model is obtained.

In chapter 5 we analyze binary relational data bases obtained using this method with respect to the three main objectives : data independance, data readability and sharability and nonredundancy of data. We give some of the characteristics of the model and compare it to the general relational model.

Appendix A contains the Pascal program for the implementation of the algorithms presented in the thesis.

Chapter II

THE MODEL

2.1 INTRODUCTION

Wihtin the context of the data structure design problem five levels of data refinement can be identified (To):

1. Data reality: the real world, data object.

2. Conceptual model: a model of the real world where only the properties relevant to the specific applications are contained.

3. Abstract structure: a refinement of the conceptual model in which only some properties are made explicit, the others derivable indirectly through computation.

4. Storage structure: a model of the abstract structure in terms of representation for each of the data type occurrences.

5. Primitive encoding: the computer representation that concerns the physical devices involved.

Associated with each level of abstraction is a set of operations performed on the data. The transition from one level of abstraction to another will not be considered, here; instead, the focus will be on the design of efficient abstract structures for data bases.

In chapter 1, the problem under consideration has been briefly and informally described. In this chapter the model and some theoretical results obtained in (Sa) are presened.

2.2 THE MODEL

2.2.1 Domains

Definition: A domain is a set of items each of the same type.

Unlike traditional definitions, we allow a domain to be more than simply a set of single values. Each occurrence in the domain, (i.e., each item) is identified by a single unique value for its type.

2.2.2 Relations

Logical relationships between the elements of different domains are called relations. The set of all relations should be known at design time; some of them will be explicitly stored, others are available through computation. In our model, only binary relations among domains are considered. Let λ denote the null element.

Definition: A binary relation r between two domains D1 and D2 is a non-empty subset of the cartesian product $D(1)\cup\{\lambda\} \times D(2)\cup\{\lambda\}$ where:

(1) $\forall d(1) \in D(1)$ $\exists d(2) \in D(2)\cup\{\lambda\}$ such that $(d(1),d(2)) \in r$

(2) $\forall d(2) \in D(2)$ $\exists d(1) \in D(1)\cup\{\lambda\}$ such that $(d(1),d(2)) \in r$

$d(1)r\lambda$ will denote the fact that d(1) is not related to any element of D(2). For sake of simplicity $(d(1),d(2)) \in r$ will be represented by d(1)rd(2).

The right side (respectively, the left side) of a relation r denoted by RS(r) (respectively, LS(r)) is defined as follows:

$$RS(r) = \{ a \in D(1) \mid \exists b \in D(2) \text{ such that } arb \}$$

$$LS(r) = \{ b \in D(2) \mid \exists a \in D(1) \text{ such that } arb \}$$

A binary relation r can then be viewed as a table with two columns and as many rows as there are tuples in the relation r, the two columns being RS(r) and LS(r).

Given a relation r between two domains D(1) and D(2), the inverse of r denoted by r^{-1} is a relation between D(2) and D(1) consisting of all tuples (d(2),d(1)) such that (d(1),d(2))ϵr. Consequently $RS(r^{-1})=LS(r)$ and $LS(r^{-1})=RS(r)$. Notice that if a relation r between D(1) and D(2) exists then r^{-1} can be derived from r.

The ability to cost-efficiently locate records in a file given any fields or keys is essential to the effective operation of today's computer-based information systems. With respect to the above model, let r be a binary relation between domains D(1) and D(2); the question (or query)is: given aϵD(1) determine the set of all bϵD(2); such that arb. Depending on r, this set can be empty (i.e. contains only λ) or non-empty . The value of the query q(r)(a), aϵD(1),is the set of all values bϵD(2) such that arb.

2.2.3 Composition

The basic operation on binary relations is the composition. Let r and r' be two binary relations between domains D(1) and D(2), and between D(2) and D(3) respectively. The composition of r and r' denoted by r*r' is the binary relation defined by:

1. $\forall(a,b)\in D(1)\times D(3)$ if $\exists c\in D(2)$ $(a,c)\in r$ and $(c,b)\in r'$ then $(a,b)\in r*r'$
2. $\forall a\in D(1)$ if $\forall b\in D(3)$ $(a,b)\in r*r'$ then $(a,\lambda)\in r*r'$
3. $\forall b\in D(3)$ if $\forall a\in D(1)$ $(a,b)\in r*r'$ then $(\lambda,b)\in r*r'$

2.2.4 Constraints

Consider a set of i domains D(1),...,D(i), and i binary relations r(1),...,r(i) among them, where r(j) is a binary relation between D(j) and D(j+1), $1\leq j<i$, and r(i) is a binary relation between D(1) and D(i). If r(i) can be derived from r(1),...,r(i-1) by a sequence of compositions of relations over common domains (i.e. r(i)=r(1)*r(2)*...*r(i-1),

where * denotes the composition operation), then a constraint is said to exist on the i relations; in particular, relation r(i) is said to be derivable from r(1),...,r(i-1) through a sequence of composition operations. Notice that the order in which the relations are composed is meaningful.

2.3 THE CONCEPTUAL MODEL AND ABSTRACT STRUCTURES

2.3.1 Conceptual model

As mentioned before, the conceptual model is a model of the real world where only the properties relevant to the specific applications are contained. Two general principles for the conceptual model, that should be observed at all times, have been stated in (Iso). These are :

100 percent principle

All relevant general static and dynamic aspects, i.e. all rules, laws etc., of the universe of discourse should be described in the conceptual schema. The information system cannot be held responsible for not meeting those described elsewhere, including in particular those in application programs.

conceptualization principle

A conceptual model should only include conceptually relevant aspects, both static and dynamic, of the universe of discourse, thus including all aspects of (external or internal) data representation; physical data organization and access as well as all aspects of particular external user representaion such as message formats, data structures, etc.

In this model, the domains are assumed to be disjoint. The properties relevant to the specific applications are those binary relations among the domains and the constraints on the relations. Let D denote the set of domains, R denote the set of relations and C the set of constraints. We can then define a conceptual model as follows:

Definition: A Conceptual model I is a triple I = (D, R, C) where D, R and C are defined as above.

2.3.2 Graph representation

It is useful to represent the set of domains and the set of relations among them of the conceptual model by a labelled directed graph (Cha, Got,Tsu,Sa). Each node in the graph represents a domain, and each edge represents a relation between two domains. If I=(D,R,C) is a conceptual model, the directed graph representing D and R is G(D,DxDxR) where D is the set of nodes of G, and the cartesian product DxDxR is the set of edges. This directed graph will be used as a mean for the automated design of binary relations and data bases. In the next chapter, it will be used to define the conceptual model.

2.3.3 The abstract structure

Definition: An abstract structure A implementing the conceptual model I = (D, R,C) is a triple A = (D,R(A),Q(A)) where:

1. D is as defined in I.

2. $(R(A))^* \supseteq R$ where $(R(A))^*$ denotes the set of all distinct relations that can be obtained through composition of elements of R(A).

3. $Q(A): R \rightarrow s(R(A))$ is a query mapping defined as follows: for all $r \in R$, Q(A)(r) is a sequence $q \in s(R(A))$ of minimum length such that q=r (i.e. q and r are equivalent.)

Given a conceptual model I, there exists at least one abstract structure implementing it. Consider the query mapping as being the identity function denoted by $\mathcal{J}$. The triple (D, R, $\mathcal{J}$) is an abstract structure implementing I.

2.4 EFFICIENCY MEASURES

2.4.1 Preliminary considerations

Given a conceptual model I=(D,R,C), a large number of abstract structures implementing it may exist. The question is, what are the properties an abstract structure should satisfy in order to be chosen.

Two factors will have a definite influence on such a decision. These are the two components of the abstract structure: the set of relations and the query mapping.

If storage space is at a premuim, an abstract structure requiring the minimum amount of storage space to store its relations should be preferred. In general, the number of tuples of any two relations may not be the same. For sake of simplicity, we will assume that all relations are of same size . In this case minimum storage implies minimum number of relations.

There may be some disadvantages in abstract structure with minimum number of relations. To answer a query, some of the unstored relations might be needed; in this case, they must be formed from stored relations by composing them. This will undoubtly increase the response time to the queries. On the other hand, if all the relations are explicitly stored, then every query will require the retrieval of only one relation; that is, every query is answered with minimum response time.

Let us assume that the retrieval of a relation takes one unit of time. It is clear from the above discussion that decreasing the number of explicitly stored relations will in general increase the response time, and increasing the number of stored relations will in general decrease the response time. The response time may be different for different queries for the same abstract structure; this is so because some of the queries require the retrieval of a larger number of relations than others. In this thesis, the response time of an abstract structure is considered as the average response time to all queries that can be possibly asked.

In the previous sections, relational graphs have beewn defined. Some useful properties of the relational graphs were identified in (Sa) and will be presented here without giving details of the proofs.

2.4.2 Definitions

First observe that the set of abstract structures and the set of relational graphs are isomorphic (Sa). That is for every abstract structure A(i) there exists a relational graph Hi representing it, and vice versa. Let F(H*) denote the set of the isomorphic images of the implementations of I.

Definition: Given a query $q \in R$ and given $H(i) \in F(H^*)$, the retrieval time for q in H(i) is defined to be:

$$Q(i)(q) = |Q(j)(q)|$$

A relational graph $H(i) \in R(H^*)$ is said to be d-time efficient if $\forall H(j) \in R(H^*)$

$$Q(i)(q) < Q(j)(q)*d$$

If L(H(i)) denote the number of arcs in H(i), the relational graph H(i) is said to be d-space efficient if for all $H(j) \in F(H^*)$:

$$L(H(i)) \leq L(Hj)*d$$

Let $H(i) \in F(H^*)$, SPACE(Hi) be the minimum & such that H(i) is &-space efficient, and TIME(H(i)) be the minimum & such that H(i) is &-time efficient.

A relational graph $H(i) \in F(H^*)$ is d-time optimal if it is d-time efficient and $\forall H(j) \in \{H(k) \in F(H^*)$ s.t. $TIME(H(k)) \leq d\}$

$$SPACE(H(i)) \leq SPACE(H(j))$$

A relational graph $H(i) \in F(H^*)$ is d-space optimal if it is d-space efficient and $\forall H(j) \in \{H(k) \in F(H^*)$ s.t. $SPACE(H(k)) \leq d\}$

$$TIME(H(i)) \leq TIME(H(k))$$

In general increasing the space efficiency will decrease the time efficiency, and decreasing the space efficiency will increase the time efficiency. The choice will depend on the particular application. In the next sections the abstract structures are classified according to these efficiency parameters.

2.4.3 Classification of abstract structures

As mentioned in the previous section, for any conceptual model there exists at least one abstract structure implementing it. In this structure, the query mapping is the identity function, and the set of relations is equal to that of the conceptual model; this structure shall be referred to as the trivial abstract structure. The trivial abstract structure is 1-time efficient; in fact all the relations are

explicitly stored, and every query required the retrieval of only one relation. That is, the average response time is minimum and is equal to one. On the other hand, the space efficiency of the trivial abstract structure is very high since all the relations are explicitly stored, requiring a maximum amount of storage space. The trivial abstract structure represents an extreme case: maximum time efficiency and minimum space efficiency. An abstract structure with opposite properties (i.e., maximum space efficiency (minimum number of explicitly stored relations) and minimum time efficiency) is difficult to determine.

Consider a conceptual model $I=(D,R,C)$ and an abstract structure $A=(D,R(A),Q(A))$ implementation of I. A is said to be a perfect implementation of I if $(R(I))^*=R$. Recall in the definition of an abstract structure it was only required $(R(I))^* \supseteq R$. In a perfect abstract structure, only the relations in R can be formed. Moreover, an implementation A is said to be a minimal perfect implementation of I iff A is a perfect implementation and $\forall r \in R(A)$ there is no abstract structure $A'=(D,R(A)-r,Q(A'))$ implementing I. That, is no proper subset $R(A') \supsetneq R(A)$ and A' exists such that

A'=(Ð,R(A'),Q(A')) is an abstract structure implementing I. Notice that more than one minimal perfect implementation of a conceptual model may exists, and that there always exists at least one. One of the advantages of this class of abstract stuctures is that they might be computed automatically; it will be shown later how. The second advantage is that they are space efficient, but yet not optimum. The reason behind the interest in this class of abstract structures is that, unlike the space optimum abstract structures, they can be easily computed. The only obvious means to determine an optimum space abstract structure (an abstract stucture with minimum number of relations) is to exhaustively search all possible abstract structures implementing the conceptual model. This is not feasible for any large data bases.

2.5 EXISTING TECHNIQUES

2.5.1 Equivalences

It has been proven that the set of abstract structures and the set of relational graphs are isomorphic (Sa). That is every abstract structure can be represented by a relational graph and every relational graph is the representation of an abstract structure. Furthermore, the problem of finding an efficient abstract structure can be reduced to the problem of finding an efficient relational graph. A methodology to find an efficient abstract structure has been defined (Sa) as follows:

1. Find a minimal perfect implementation A(i) of I.
2. Derive the natural graph H(i).
3. Find in F(H(i)) a graph H(j) that minimizes certain costs.
4. Derive the abstract structure I(j) having H(j) as its natural graph.

Steps 2 and 4 are easy to perform. In this chapter, it is shown how to perform steps 1 and 3. Consider step 1; that is, finding a minimal perfect implementation A(i) of I.

An algorithm to construct a minimal abstract structure and a proof of correctness have been in (Sa). The algorithm works recursively. It takes an element of the set of constraints and try to add it to the query mapping. In order to add an element to the query mapping the algorithm search all its previous elements. An element would be added to the query mapping after checking that the substitution of any occurrence of the left side of the new element in any element of the query mapping by the right side of that element does not create a cycle. Otherwise, this element cannot be added to the query mapping. The query mapping is then formed after processing all the elements of the set of constraints. The set of relations of the minimal perfect implementation is composed of those relations which do not appear as a left hand side member of an element of the query mapping.

2.5.2 Hierarchical conceptual model

In chapter 2, the concept of a constraint and cycle in a graph have been defined; a description of how to determine the set of constraints by checking all the cycles of the graph has been given; and it will be shown that not every cycle leads to a constraint. In particular, if r(1),...,r(n) are the edges in the cycle, it is possible that

$\nexists$ i$\in${1,...,n} s.t. r(i)=r(i+1)*..*r(n)*r(1)*..*r(i-1)

If this is not the case (that is,if for every cycle r(1),...,r(n) there exists i $\in$ {1,...,n} such that r(i)=r(i+1)*...*r(n)*r(1)*..*r(i-1)) we will say that the conceptual model is hierarchical.

It is this class of conceptual models that will be considered in the next sections. It has been proven (Sa) that if I is a hierarchical conceptual model then the following properties hold:

1. there exists at least one minimal perfect implementation.
2. h(H0) is a tree
3. H0 is 1-time efficient

4. H0 is (n-1) time efficient

5. H* is m/n-1 -space efficient

6. H* is 1-time efficient

Later on, it will be made use of properties (1) and (2) to find an efficient abstract structure implementing a hierarchical conceptual model.

2.5.3 The redundant graph

In the previous section the model and efficiency parameters have been defined, and an algorithm to construct an abstract structure which is space efficient has been described: the minimal perfect implementation. In constructing this abstract structure, the goal was to delete, from the original set of relations, as many relations as possible provided that the resulting abstract structure

still implements the original conceptual model. In other words, only the space efficiency of theabstract structure was of concern, regardless of the time efficiency; consequently, the time efficiency in this structure may degrade.

The main idea behind the Redundant graph solution is that, by carefully adding relations to the minimal perfect implementation, the resulting abstract structure is guaranteed to implement the original conceptual model with constant response time and $(n-3)\log n + n - 1$ explicitly stored relations. Before describing the main steps of the algorithm, some definitions will be introduced.

Let G(V,E) be a tree where V is the set of vertices and E the set of edges. Let $x \in V$, and G(V(x),E) denote the directed tree rooted in x. Let x(1),...,x(k) be the children of x in V(x); and let g(x(i)) denote the number of vertices in the largest subtree of G(V(x),E) rooted at x(i) not containing a, and denoted G(V(x,x(i)),E(x(i))). Without loss of generality, assume g(x(1))>g(x(2))>...>g(x(k)). Define:

$$\text{MaxUnb}(x) = g(x(1)) - g(x(n))$$

$$\text{RelUnb}(x) = g(x(1)) - g(x(k))$$

$$Unb(x) = Min(MaxUnb(x), RelUnb(x))$$

Node x is said to be a best partition in V if $\forall y \in V$, $Unb(x) \leq Unb(y)$. The algorithm can be described as follows:

1. for every $x \in G$ find Unb(x)
2. determine t=min Unb(x)
3. for all direct children x(1),...,x(k) of x form the subtree rooted at x(i) and go to step 1 with x=x(i), G(V(x,x(i)),E(x,x(i)))
4. connect t to all $y \in V(x)$ where y is not a direct child of x

This algorithm have been implemented and coded in Pascal programming language. The code appears in Appendix (A). Boolean matrices are used to represent a graph: entry (i,j) is 1 if and only if there is a relation between domains i and j.

2.5.4 The query mapping

The objective of the redundant graph construction is to construct an abstract structure which needs O(nlog n) explicitly stored relations and a maximum response time of 2. It can be easily seen that it is always possible to go from any node in the redundant graph to any other by traversing at most 2 edges. In order for the abstract structure to guarantee a maximum respose time of 2, all the relations corresponding to the edges added by the procedure should be explicitly stored, in additon to the original graph. Yet another component needs to be defined: the query mapping. This is the mean by which any path in the original graph can be transformed into a path of length at most 2 in the new graph. The query mapping can be defined as follows:

```
for every subtree V(x) with x as best partition node do
 for every child x(i) of x do
  for every y∈V(x,x(i)) do
   for every z y,x z∈V(x) do
    If a relation r(y,z) relevant to the application exists
        then do
     If z∈V(x,x(i)) then replace previous value
        of r(y,z) by r(y,z) = r(y,z)*r(x,z)
     end if
    end if
```

Chapter III

EFFICIENT IMPLEMENTATIONS.

3.1 INTRODUCTION

In the previous chapter the binary relational model and the concept of constraints have been presented. Also, some theoretical results obtained in (Sa) has been given. In this chapter some practical implications of these theoretical results will be considered. It is shown that, with the introduction of the concept of constraints as a component of the conceptual model, the integrity problem, (which is one of the complex problems of relational data bases) is avoided. This will be illustrated with an example. It is also shown

how to practically implement the theoretical results presented in chapter 2. A procedure for defining the conceptual model is given, and the implementation of the minimal perfect implementation and of the redundant graph is illustrated by examples.

3.2 THE INTEGRITY PROBLEM

An important problem in a data model concerns the integrity of the system; that is, the property of obtaining correct answers. To illustrate this concept, we use an example given in Cardenas (Ca). Figure 3.1 (a),(b),(c) shows three relations : EMPLOYEE, EXPER and EMP-EXP. Joining the two relations EMPLOYEE and EXPER over the domain EXPERTISE we obtain the relation EMP-EXP' shown in Fig. 3.1.(d). This relation contains the actual information of EMP-EXP but it also contains additional information which is incorrect (e.g. — THOMAS is in CHICAGO) and is indistinguishable from correct information. In our model, integrity is always preserved: operations over relations can be performed only according to a predefined set of explicit contraints among the relations.

EMPLOYEE

EMP#	E-NAME	EXPERTISE	AGE
1	JONES	ACCOUNTING	30
2	MARTIN	COMP-SCI	35
3	GOMEZ	MGT-SCI	42
4	CHU	COMP-SCI	34
5	DAMES	ACCOUNTING	40
6	THOMAS	ACCOUNTING	52
7	WATSON	MGT-SCI	39

(a)

EXPER

EXPERTISE	LOCATION
ACCOUNTING	CHICAGO
COMP-SCI	LOS ANGELES
MGT-SCI	LOS ANGELES
MGT-SCI	MEXICO CITY
ACCOUNTING	LONDON

(b)

EMP-EXP

EMP#	E-NAME	EXPERTISE	AGE	LOCATION
1	JONES	ACCOUNTING	30	CHICAGO
2	MARTIN	COMP-SCI	35	LOS ANGELES
3	GOMEZ	MGT-SCI	42	MEXICO CITY
4	CHU	COMP-SCI	34	LOS ANGELES
5	DAMES	ACCOUNTING	40	CHICAGO
6	THOMAS	ACCOUNTING	52	LONDON
7	WATSON	MGT-SCI	39	LOS ANGELES

(c)

EMP-EXP'

EMP#	E-NAME	EXPERTISE	AGE	LOCATION
1	JONES	ACCOUNTING	30	CHICAGO
1	JONES	ACCOUNTING	30	LONDON
2	MARTIN	COMP-SCI	35	LOS ANGELES
3	GOMEZ	MGT-SCI	42	LOS ANGELES
3	GOMEZ	MGT-SCI	42	MEXICO CITY
4	CHU	COMP-SCI	34	LOS ANGELES
5	DAMES	ACCOUNTING	40	CHICAGO
5	DAMES	ACCOUNTING	40	LONDON
6	THOMAS	ACCOUNTING	52	CHICAGO
6	THOMAS	ACCOUNTING	52	LONDON
7	WATSON	MGT-SCI	39	LOS ANGELES
7	WATSON	MGT-SCI	39	MEXICO CITY

(d)

Figure 3.1

3.3 CONSTRUCTION OF THE CONCEPTUAL MODEL

3.3.1 Example

Let us consider again the example of EMPLOYEE-EXPERTISE-LOCATION illustrated in the previous section, and assume that the basic queries which will be asked are the following:

1- For a given employee find his location.
2- For a given employee find his experise.
3- For a given expertise find its location(s).
4- For a given experitise find employee(s).
5- For a given location find expertise located there.
6- For a given location find employee(s).

Let us first observe that the previous representation does not conform to our model: firstly, in our model all rela-

tionships among data are binary relations among domains; secondly, elements of the domains are unique; finally each value may represent an instance of a record type. To be consistent with our model we then define a new set of record types as follows:

EMPLOYEE

EMP#	E-NAME	AGE
1	JONES	30
2	MARTIN	35
3	GOMEZ	42
4	CHU	34
5	DAMES	40
6	THOMAS	52
7	WATSON	39

EXPERTISE

EXPER
ACCOUNTING
COMP-SCI
MGT-SCI

LOCATION

CITY
CHICAGO
LOS ANGELES
MEXICO CITY
LONDON

Figure 3.2

3.3.2 Construction of the conceptual model

The process of defining the conceptual model corresponding to the real world is not an easy task. The designer should have a thorough understanding of the model, of all actual and future applications and have some statistics regarding the usage of the data. The inclusion of a property or a relation irrelevant to the specific applications unnecessarily complicates the model and wastes space to store it; on the other hand, the model would be inefficient if some important properties of the data were not identified in the conceptual model because of misinformation about the usage of the data. The other component of the conceptual model which needs to be defined with special care is the set of constraints. In a previous section it has been shown through an example how incorrect informations can be derived by joining relations in absence of constraints. This emphasizes the role that the constraints play to maintain integrity information. The designer of the conceptual model should

check for every constraint of the form r=r(1)*...*r(i) whether in fact the join of relations r(1),r(2),...,r(i) is equal to the relation r. It might happen that this is true only for some particular instances of the data. In such a case r=r(1)*...*r(i) can not be considered as a constraint.

3.3.3 Relations

When constructing the conceptual model, after identifying the domains, the next step is to identify the relations among these domains and define them. At this stage, one can check whether the set of relations can satisfy all queries which might be asked. This can be done by considering each domain in the graph and ensuring that every relation involving that domain is represented by an edge; if this is not the case, the relation corresponding to the missing edge should be added to the set of relations. The next step is to define the set of constraints on the relations.

Define the domain EMP# with values

1, 2, 3, 4, 5, 6 and 7,

each of these values identifying a unique record, the domains EXPER and CITY are defined as before.

Let us also define the set of relations:

```
EMPLOYEE-EXPERTISE ={ (1,ACCOUNTING), (2,COMP-SCI),
                      (3,MAG-SCI), (4,COMP-SCI),
                      (5,ACCOUNTING), (6,ACCOUNTING)  }
                      (7,MGT-SCI)

EMPLOYEE-LOCATION = {(1,CHICAGO), (2,LOS-ANGELES),
                     (3,MEXICO), (4,LOS-ANGELES), (5,CHCAGO),
                     (6,LONDON), (7,LOS-ANGELES) }

EXPERTISE-LOCATION = {(ACCOUNTING,CHICAGO),
                      (COMP-SCI,LOS-ANGELES), (MGI-SCI,LOS ANGELES),
                      (MGI-SCI,MEXICO-CITY), (MGI-SCI,MEXICO-CITY),
                      (ACCOUNTING,LONDON) }
```

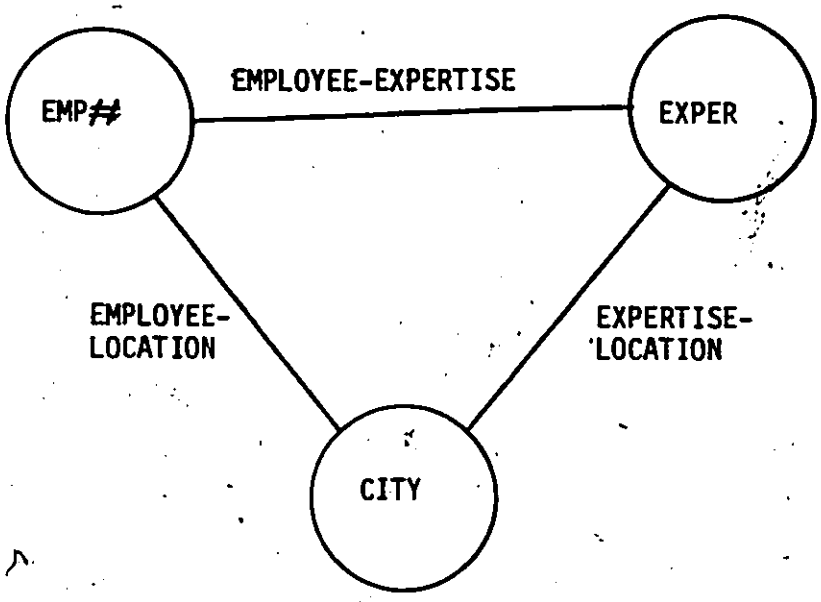

Figure 3.3

The graph representing these sets of domains and relations is shown in figure 3.3. The relations can also be represented by the tables of figure 3.4.

EMPLOYEE-LOCATION

EMP#	CITY
1	CHICAGO
2	LOS ANGELE
3	MEXICO CITY
4	LOS ANGELES
5	CHICAGO
6	LONDON
7	LOS ANGELES

(a)

EMPLOYEE-EXPERTISE

EMP#	EXPER
1	ACCOUNTING
2	COMP-SCI
3	MGT-SCI
4	COMP-SCI
5	ACCOUNTING
6	ACCOUNTING
7	MGT-SCI

(b)

EXPERTISE-LOCATION

EXPER	CITY
ACCOUNTING	CHICAGO
COMP-SCI	LOS ANGELES
MGT-SCI	LOS ANGELES
MGT-SCI	MEXICO CITY
ACCOUNTING	LONDON

(c)

Figure 3.4

3.3.4 A procedure for defining the constraints

The next step in the construction of the conceptual model, is the definition of the constraints. The corresponding

graph can present a great help for this purpose. For every cycle in the graph we must check whether a constraint can be derived. This can be done by executing the following steps on the cycle.

For all cyclic permutation of the nodes in the cycle:

1. Number the nodes on the cycle from 1 to n (where n is the number of nodes in the cycle).
2. Write down the relation corresponding to r(1,2) (r(1,2) denotes the relation between 1 and 2).
3. Join all relations corresponding to the edges (i,i-1), i=2,...,n, and (1,n) on the common domains. (recall that if r(i,i-1) is in the set of relations, r (i-1,i) is also in the set).
4. Check if relation derived in step 2 and relation derived in step 3 are equal.
5. If yes then there is a constraint r(1,2)=r(1,n)*r(n,n-1)*..*r(3,2) otherwise there is no such constraint.

In the previous example we showed the graph contains one cycle, which contains three edges. Joining the relations EMPLOYEE-EXPERTISE and EXPERTISE-LOCATION on the domain EXPER gives a relation with LS(R)=EMP# and RS(R)=CITY and is shown in fig.(3.5) (a).

r

EMP#	CITY
1	CHICAGO
1	LONDON
2	LOS ANGELES
3	LOS ANGELES
3	MEXICO-CITY
4	LOS ANGELES
5	CHICAGO
5	LONDEON
6	CHICAGO
6	LONDON
7	LOS ANGELES
7	MEXICO CITY

(a)

r'

EXPER	CITY
ACCOUNTING	CHICAGO
COMP-SCI	LOS-ANGELES
MGT-SCI	LOS ANGELES
MGT-SCI	MEXICO-CITY
ACCOUNTING	LONDON

(b)

r"

EMP#	EXPER
1	ACCONTING
2	COMP-SCI
2	MGT-SCI
3	MGT-SCI
4	COMP-SCI
4	MGT-SCI
5	ACCOUNTING
6	ACCOUNTING
7	COMP-SCI
7	MGT-SCI

(c)

Figure 3.5

Relation r is derived by joining EMPLOYEE-EXPERTISE and EXPERISE_LICATION. It is obvious that r is not equal to EMPLOYEE-LOCATION relation and no constraint can be derived. On the other hand joining relations $(EMPLOYEE\text{-}EXPERTISE)^{-1}$ with EMPLOYEE-LOCATION on the domain EMP# gives relation r' shown in figure 3.5 (b) with LS(r)=EXPER and RS(r')=CITY and this relation is equal to the relation EXPERTISE-LOCATION. Hence there is a constraint $EXPERTISE\text{-}LOCATION=(EMPLOYEE\text{-}EXPERTISE)^{-1}*EMPLOYEE\text{-}LOCATION$. The relation r'' with LS(r'')=EMP#, RS(r'')=EXPER in figure 3.5 (c) is not equal to EMPLOYEE-EXPERTISE and no constraint is derived.

At this point we have shown how to define the three components of the conceptual model. That is the domains, the relations and the constraints. In the following we will go one step further and show how to efficiently implement a conceptual model.

3.4 CONSTRUCTION OF ABSTRACT STRUCTURES

Back again to the example, two abstract structures implementing the conceptual model defined abovecan be defined. These are the abstract structure A = (D, R(A), $\mathcal{J}$) where $\mathcal{J}$ is the identity function,

D={EMP#, EXPER, CITY};

R(A) = { EMPLOYEE-EXPERTISE, EMPLOYEE-LOCATION, EXPERTISE-LOCATION};

and the abstract structure A'={D, R(A'), Q(A') where D is as above, R(A')= {EMPLOYEE-EXPERTISE, EMPLOYEE-LOCATION}, and Q defined by

Q(EMPLOYEE-EXPERTISE) = EMPLOYEE-EXPERTISE

Q(EMPLOYEE-LOCATION) = EMPLOYEE-LOCATION

Q(EXPERTISE-LOCATION) = (EMPLOYEE-EXPERTISE)$^{-1}$ * EMPLOYEE-LOCATION.

Notice that for each relation included in the set of relations, it is implied that its inverse is also included.

As for the conceptual model, the set of domains and the set of relations can be represented by a graph. The graph corresponding to the abstract structure A is the same as for

the conceptual model. Figure 3.6 shows the graph corresponding to A'.

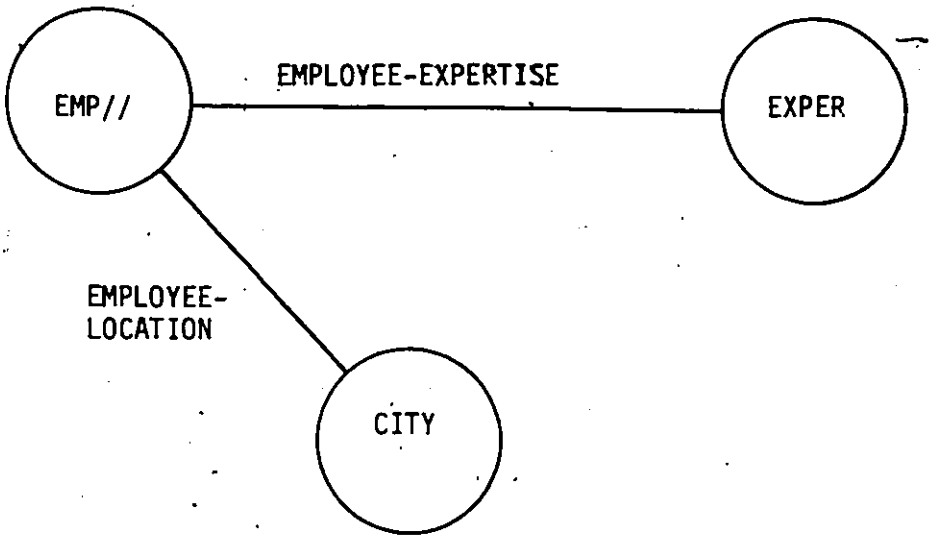

Figure 3.6

3.5 THE MINIML PERFECT IMPLEMENTATION AND THE REDUNDANT GRAPH

In chapter 2, the procedure for constructing a minimal perfect implementation has been described. The inputs of the procedure are the names of the relations and the constraints among them. The output is the set of relations to be explicitly stored, and the query mapping. Some comments are

in order. When describing the procedure for constructing the redundant graph, we has been assumed that the information about which domains are related by each relation is provided. This is necessary since, to directly go from one node to another, vit is necessary to know whether an edge connecting them exists (i.e., whether a relation among them exists). Also, after adding an edge between two nodes, it is necessary to know what is the name of the relation relating the domains corresponding to the nodes. Therefore, in order to go from the minimal perfect implementation to the redundant graph two additional pieces ofinformation must be supplied:

1. For each pair of domains whether they are directly connected or not. This can be provided as a boolean matrix as mentioned before.
2. For ecach relation r, what is the pair of domains related by r. This can be defined by an nx3 matrix where n is the number of relations. The entries of the first column are the name of the relation, the 2nd and 3thd is the first and 2nd domains.

3.6 SUMMARY OF THE PROCESS

The whole process of efficient abstract structure definition can be summarized as follows

1. Define the conceptual model, namely the domains, relation among them and the constraints.
2. Execute the "the minimal perfect implementation procedure" with: name of the relations and the constraints as input. The output is a minimal perfect implementation: The domains (which are the same as in the conceptual model), the name of the relations to be explicitly stored , and the query mapping.
3. If response time is not very important, then stop at this stage, otherwise continue.
4. Supply the following two informations:
   a) for each pair of domains is there a relation relevant to the application.
   b) for each relation r what is the pair of domains related by r.

5. using the information in H, transform the set of explicitly stored relations of the minimal perfect implementation in 2 into a graph which is a tree.
6. Execute the "Redundant graph procedure". This will output a new graph, whose edges corresponds to the set of explicitly stored relations; it will also update the query mapping.
7. For each relation added by the procedure "redundant graph", check if this relation is relevant to the applications. If yes do nothing, ohterwise check in the query mapping if this relation appears in the definition of another relation. If so do nothing, otherwise delete it from the set of explicitly stored relations.

3.7 COMPARISON WITH THE GENERAL RELATIONAL MODEL

Compared to the general relational model, the binary relational model has both advantages and disadvantages. The major disadvantage is that relations cannot be defined dynamically as in the general relational model; in fact, relations are defined only at design time. However this fact has also its advantages. Namely, there is no need for a data definition language as in the relational model. Furthermore, several problems which need careful and sometimes very complex solutions are avoided (e.g., the referential integrity, wrong functional dependencies, etc.) The referential integrity problem is the problem of ensuringthat all values of a domain appear in at least one tuple of a relaion. The wrong functional dependencies problem have been explained before. Avoiding these problems means reducing the amount of software needed to implement the model. For the model presented here, what is needed is an information retrieval system. An efficient abstract structure can be automatically designed as shown before. The degree of control of the efficiency in general relational model is smaller, and this is

based only on observations of the designer and no measure is provided.

Chapter IV

A DETAILED EXAMPLE

4.1 INTRODUCTION

Consider an academic database, defined as follows. The faculty of science and engineering wants to computerize the retrieval of information regarding its departments, the professors, the students, their backgroundsand the courses offered by each department. The faculty has severalprofessors teaching courses, several students, and offers several courses. The following assumptions hold: each professor of the faculty is a member of only one department, and teaches at least one course offered by his/her department. Students come from different places and different programs. They have different backgrounds.

4.2 THE QUERIES

Occasionnally, the following information needs to be retrieved from the database:

1 . Which are professors in department x?

2 . Which are the courses offered by department x?

3 . Which are the students of department x?

4 . What is the background of the students taking courses in department x?

5 .Which courses are taught by professor x?

6 . To which department does member professor x belong?

7 . Which students are taking courses from professor x?

8. What are the backgrounds of the students of professor x in course y?

9. Which department is offering course x?

10. Who is the professor teaching course x?

11. Who are the students taking course x?

12. What are the backgrounds of the students taking course x?
13. Who are the professors of student x?
14. What course(s) is student x taking?
15. In what department is a student x?
16. What are the backgrounds of the student x?
17. Which students have a background x?
18. In which courses are the students who have background x?
19. Who are the professors of students having background x?
20. In which departments are the students with background x?

4.3 CONCEPTUALIZATION OF THE SYSTEM

It is clear from above that the data base contain five different types of entities. These are the professor entity, the student entity, the course entity, the department entity and the background entity. Several relationships among these entities exist. A professor is related to a department, to courses, to students and to backgrounds. A course is related to the professor who teach it, to the department which offers it, to the students who are taking it and the their background. A department is related to its professors, its students and their background and to the couses it offers. student are related to their professors, their department, their courses and background. A particular background is related to the students who have this background, the courses which contain students with this background, the departments having students with this background, and the professors teaching students with this background.

This corresponds to five domains: PROF, STUDENT, COURSE, BCKGRND and DEPT. The relationships among them can be ex-

pressed by binary relations. The graph of figure 4.1 represents the donmains and the relations among them.

The relations are:

A = DEPT-PROF

B = PROF-COURSE

C = COURSE-STUDENT

D = STUDENT-BCKGRND

E = DEPT-BCKGRND

F = DEPT-STUDENT

G = DEPT-COURSE

H = PROF-BCKGRND

I = STUDENT-PROF

J = COURSE-BCKGRND

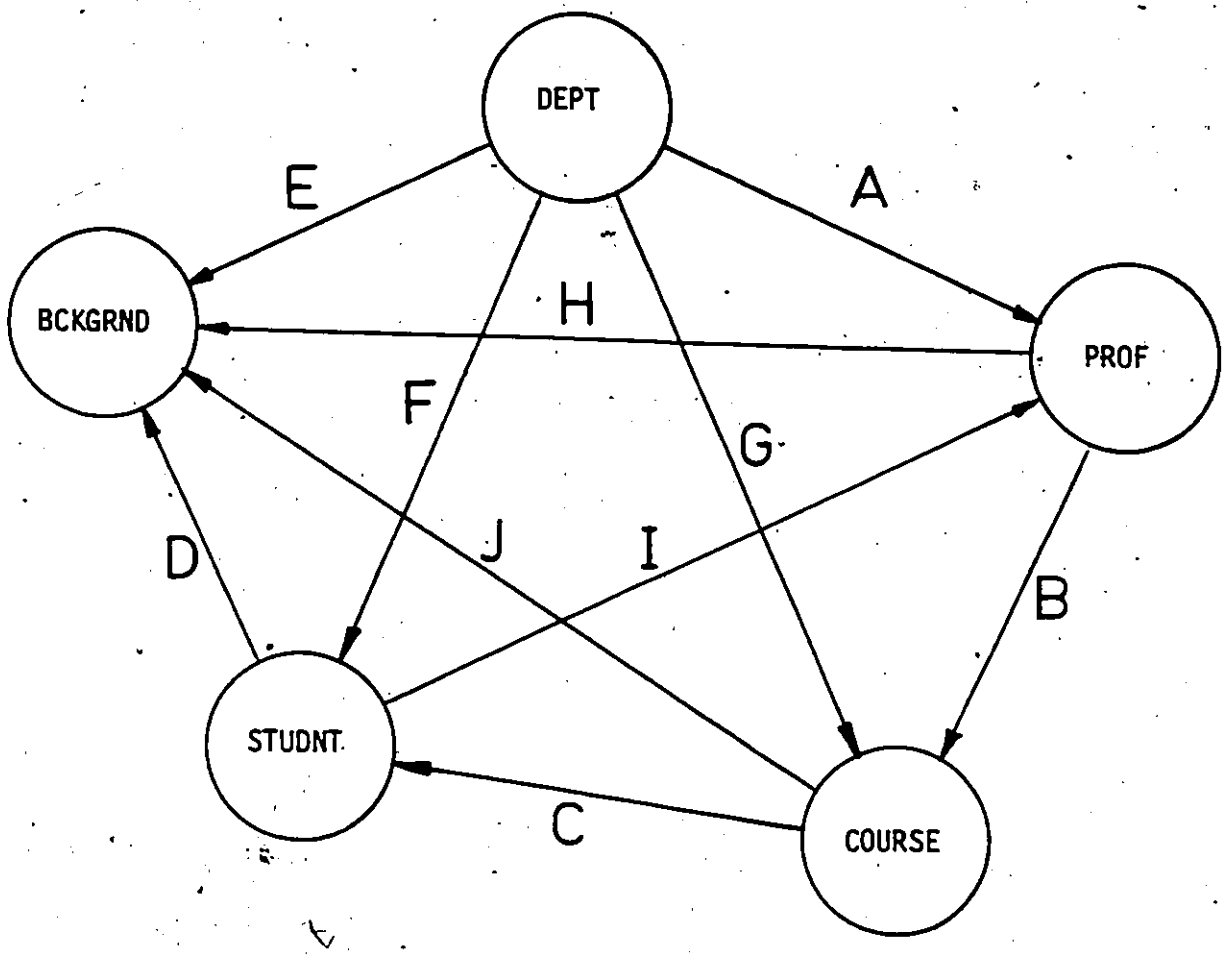

Figure 4.1

Notice that a relation exists between every two domains, and there is no need to check whether a relation is missing. A^{-1}, B^{-1}, C^{-1}, D^{-1}, E^{-1}, F^{-1}, G^{-1}, H^{-1}, I^{-1}, and J^{-1} are the inverse relations. Using these relations, all the queries of the previous section can be answered. The relations can be represented in such a way that an entry in a relation represent only the fact that an instance of a domain is related to another instance of another domain. For instance relation C can be represented by matrix, the entries of the matrix containing the marks of the students in the courses. With this representation of relation C a new query can be answered in addition to to the original one (who are the students taking course x?), that is: what are the marks of the students taking course x.

It is clear that with this set of relations as defined above we can answer all the queries that can be asked on the data base, and there is no need to enumerate each query and the corresponding relation.

In order to define the conceptual model of this data reality the set of constraints must be defined. This can be

done using the procedure described in chapter 2. The set of relations defined above have the following set of constraints:

1. $G = A*B$
2. $F = G*C$
3. $I = C^{-1} * B^{-1}$
4. $J = C*D$
5. $H = B*C*D$
6. $H = B*J$
7. $E = F*D$
8. $E = G*C*D$

And the conceptual model is:

$$C = (D,R,C)$$

D,R and C are as defined above.

4.4 THE MINIMAL PERFECT IMPLEMENTATION

In order to find the minimal perfect implementation using the program shown in appendix (A), the relations should be numbered by a sequence of integer numbers from 1 to N (N is the number of the relations in the conceptual model). We number the relations of the conceptual model as follows:

N(A) = 1, N(B) = 2, N(C) = 3, N(D) = 4, N(E) = 5, N(F) = 6, N(G) = 7, N(H) = 8, N(I) = 9, N(J) = 10

Using this notation the constraints become:

7 = 1*2

6 = 7*3

9 = -3*2

10 = 3*4

8 = 2*3*4

8 = 2*10

5 = 6*4

5 = 7*3*4

The program was run with this data as input. It produced the set of explicitly stored relations and the query mapping of a corresponding minimal perfect implementation and this is shown in page (66). Figure 4.2 shows a graph representing this minimal perfect perfect implementation. The graph is a tree, which means C is a hierarchical conceptual model. The redundant graph can then be found. The procedure TIMIMP is executed with input the graph as produced by procedure SPACOPT, after the additional input regarding relations and domains are entered, as explained in chapter 3 and in the documentation of the programs. The output of the procedure is the redundant graph of Figure 4.3.. It corresponds to the query mapping and the set of relations as printed by the procedure.

THE MINIMAL PERFECT IMPLEMENTATION CORRESPONDING

TO THE POLLOWING SET OF RELATIONS AND QUERY MAPPING

THERE ARE 4 EXPLICITLY STORED RELATIONS

THESE ARE

R 1, R 2, R 3, R 4,

THE QUERY MAPPING IS

R 7 = R 1* R 2

R 6 = R 1* R 2* R 3

R 9 = R-3* R-2

R10 = R 3* R 4

R 8 = R 2* R 3* R 4

R 5 = R 1* R 2* R 3* R 4

THE ABSTRACT STRUCTURE CORRESPONDING TO REDUNDANT GRAPH IS

THERE ARE 6 EXPLICITLY STORED RELATIONS

THESE ARE

R 1,,R 2, R 3, R 4, R-7, R10,

THE QUERY MAPPING IS

R 6 = R 7* R 3

R 9 = R-3* R-2

R 8 = R 2* R10

R 5 = R 7* R10

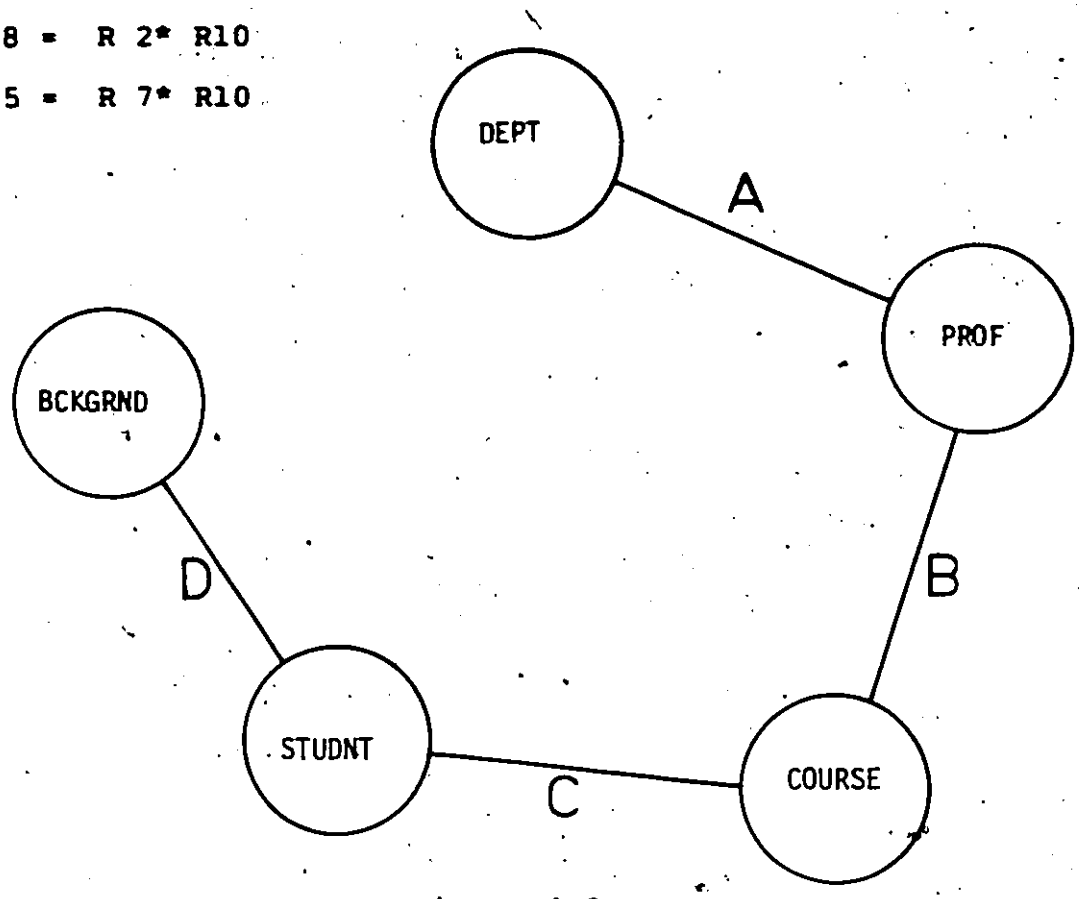

Figure 4.2

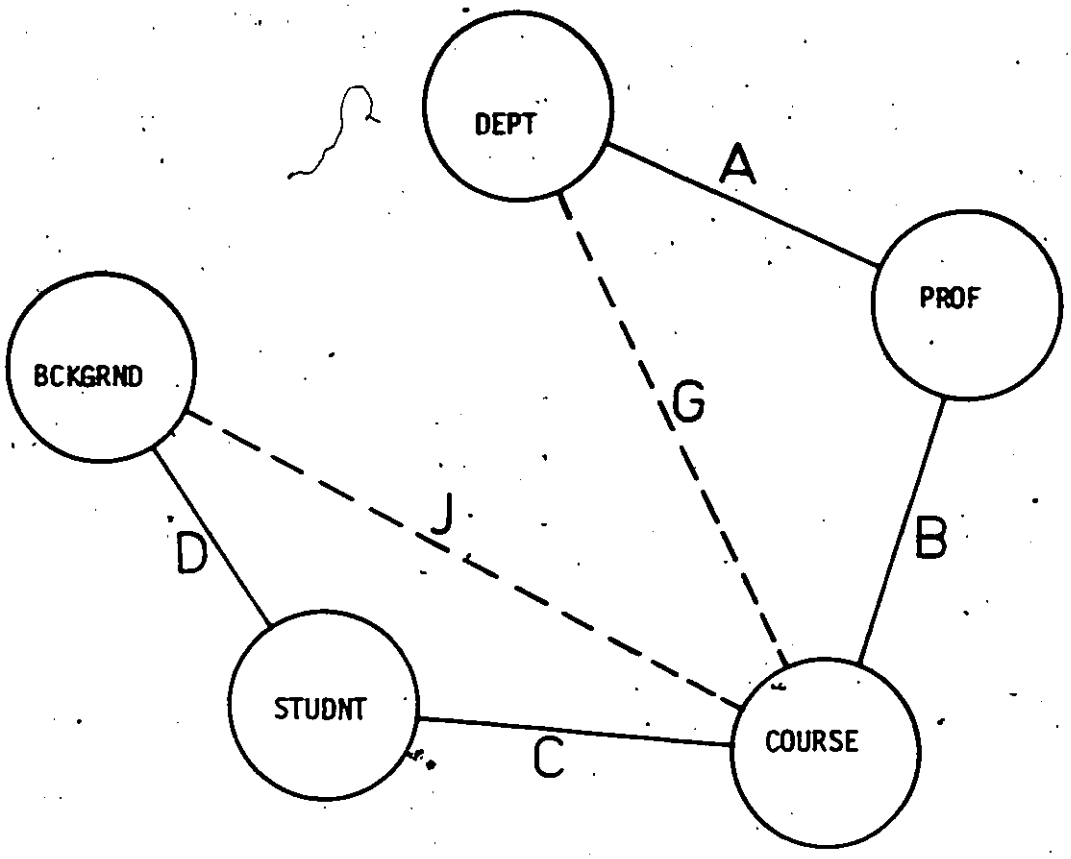

Figure 4.3

Chapter V

ANALYSIS AND CONCLUSIONS

5.1 ANALYSIS

In this section the data base obtained by applying the proposed method is discussed with reference to the main objectives of data bases: data independence, sharability and nonredundancy and relatability.

5.1.1 Data independence

The concept of data independence is central in data bases; it denotes indepencdence or insulation of application program users from a wide variety of changes in the specific logical, physical, and storage organization of the data base, Under the the proposed model, the user is unaware of how the relations are formed to answer a query. All the user need to know is the set of queries (relations) which can be

asked, not how they are answered (i.e., whether the query is answered by scanning an explicitly stored relation or by joining other explicitly stored relations). The query mapping will provide this information. We are assuming here that an interface between the end user and the data base exists. The logical design is unknown to the application programs. This implies also that any changes in the abstract structure is unimportant to the user, and gives the designer an opportunity to make changes in the logical design of the abstract structure without the need for changes in the application programs. This property of the model is very important. In fact, the content of the data may vary in time; because of this, the size of the relations can be affected, leading to large difference in size between relations. In this case, it may be convenient to store smaller size relaions rather than larger ones, at the cost of modifying the query mapping accordingly.

The relations themselves can be defined in different ways, (e.g. linked lists, Boolean matrices, etc.). The application programs are independent of the physical and of the storage organization of the relations. Hence, the data

independence (which is one of the primary goals in data base design) is achieved to a great degree in our model. Still, application programs (as in any other model) depend on the contents of the data base: in order to addres a query of the form "what are the values in domain D(2) which are related to value 'a' in D(1) ?" we need to know that domain D1 contain the value 'a', and that a relation between D(1) and D(2) exists.

5.1.2 shareability and nonredundancy of stored data

The goal is to enable applications to share an integrated base containing all the needed data, thus eliminating as much as possible the need to store redundant data. This is achieved at two levels. First, domains related to more than one domain need not be stored more than once; in this way a domain might be shared by more then one relation. Second, the relations and query mapping are shared by all the users

of the data base. Since the abstract structure implements the conceptual model, which contains all properties of the data relevant to the specific applications, all the users or applications queries' can be answered through the abstract structure. However, as seen before, some data redundancy was necessary for enhancing data base performance in terms of access.

5.1.3 Relatability

Relatability is the ability of defining relationships between records or entities at the logical level just as conveniently as defining the records themselves. The binary relational data base approach views the logical data base as a simple collection of two-dimensional tables called binary relations. These tables are easily understood and can be identified by designer and involve no consideration of access path aspects as in other models such as, hierachical ar

network. The fact that only binary relations are considered reduces the complexity of the process of determining the set of binary relations, but at the same time increase the number of relations.

5.1.4 The interface with the future

In practice, some data are referenced very frequently while others only occasionally. It is desirable to store the frequently referenced data in such a manner that they can be accessed quickly and conveniently. A major assumption was made when constructing the abstract stucture; that is, all relations are equally likely to be referenced. If this assumption doesn't hold in practice, it is always possible to explicitly store a relation which is more frequently referenced, if that may improve the performance of the data base. At the same time, if an explicitly stored relation is seldom referenced, and it is feasible to derive this relation by explicitly storing other relations which are referenced more often, this can always be done after the data base is running, without affecting the application programs. This means

that the physical organization of the data , or the abstract structure can be altered in the future conveniently. Yet, alteration at the logical level may not be possible without altering the application programs. If one of the relations, for example, becomes unimportant or meaningless in the future, deleting it means that no reference to that relation is possible anymore. Therefore applications programs must be changed; also the relations referencing it should be changed, that the query mapping should be redefined. The other possibility of adding a new relation do not affect the existing programs.

The other component which may change in the future is the set of constraints. In such a case the query mapping should be redefined. But again the end user is unaware of this fact. Redefining the query mapping may require reconstruction of the abstract structure or simple modifications of the existing one to accomodate the changes. Change in the physical storage of data, have no effect on the end user.

Summarizing, changes in the data on the conceptual level may affect the end user only in the case of relation dele-

tions; any changes on the logical or physical organization have no effect on the end user.

5.2 CONCLUSIONS

5.2.1 Summary of results

The thesis focused on the automation of the design of data bases. Some theoretical results obtained in (Sa) have been considered. It was shown that this problem can be efficiently solved. A software package coded in Pascal were developped in order to design a binary relational data base. The program takes a set of relations and the set of constraints among them and processes them according to the algorithm shown on page 15. and produces the query mapping and the set of relations of the minimal

perfect implementation. Then it process the minimal perfect implementation according to the algorithm on page 31. and

produces the set of relations of the redundant graph. This showed that the automation of the design of data bases is not just a theoretical result but can be practically implemented.

It was shown in chapter 3 how the conceptualization of the real world can be done. A procedure for the determination of the constraints was given. We observed that the concept of constraints was essential in order to avoid the integrity problem. This was illustrated by an example. Also it was showed how practically the design can be done. The binary relational model was analyzed with respect to data independence, shearability and nonredundancy and relatability. A detailed example that illustrates all the phases of the design were given in chapter 4.

5.2.2 Final remarks

Conceptualization of the real world is normally a complex task to perform. It has been noticed that using the binary relational model, this task becomes easier. Also with the introduction of the concept of constraints, the integrity

problem is avoided. The constraintsgave a means for avoiding the storage of some relations that can be derived through composition of others. On the other hand, the query mapping formed after processing the constraints will give a unique path from any node to any other in the graph. As consequence of that, the user need not to be aware of how to navigate in the graph from one node to another. In the general relational model and the network model, the user wastes a considerable amount of time to configure his path since there may be several of them that connect two nodes.

A weak point in this model is the fact that the automation can be done only when the minimal perfect implementation is a tree. The redundant graph uses that fact to determine the balanced subtrees of a tree. Some experiments are needed in order to determine the importance of this problem. It is not known whether in real applications models would tend to have the structure of a tree. It is believed that this may not be the case. Some investigation is needed to determine if a non-tree structure can be converted to a tree.

REFERENCES

Ca Alfonso F. Cardenas, Data Base Management systems, Ally and Bacon, Inc., 1979

cha L.C. Chang, D.K. Pradhan, A graph structural approach for the generalization of data management systems, Information sciences 12 (1977).

CO1 E.F. Codd, A relational Model of Data for large shared data banks, Communications of the ACM, Vol. 13, No. 6, June 1970, pp. 377-387.

CO2 E.F. Codd, Normalized data base structure: A brief tutorial, Proceedings, 1971, ACM SIGFIDET Workshop on data description, Access and control, San Diego, California, November 1971.

CO3 E.F. Codd, relational completness of data base sublanguages, in data base systems, edited by R. Rustin, Vol. 6 of the Courant computer science symposia, Printice hall, Englewood Cliffs, N.J., 1972

CO4 E.F. Codd, A Data Base Sublanguage Based on the Relational Calculus, Proceedings, 1971 ACM SIGFIDET Workshop on Data

Description, Access and Control, San Diego, California, November 1971

Got C.C. Gotlieb, A.L. Furtado, Data schemata based on directed graphs Int. j. comput. Inf. sc. 8 (1977)

Da C.J. Date, An introduction to database systems, 3rd edition, Addison-Wesley, Reading, 1980.

HAM Hammer, Data Abstractions for data bases, Proc. Conference on data abstraction Definition and Management, 1976, 58-59.

ISO ISO TC97/SC5/WG3, Concepts and terminology for the Conceptual schema and the information base, edited by JJ. Van Griethuysen

LOW J.R. Low, Automatic data structure selection : An example and overview, CACM 21 (1978)

MA J. Martin, data base organization, Printice-Hall, 1977, 2nd edition.

MIt Michael F. Mitoma, Keki B. Irani, Automatic data base shema Design and Optimization, IBM Research Report, December 18, 1975, RJ16

NE E.J. Newhold, Formal properties of data bases, In foundations of computer science, DeBakker ed., Mathematisch, Amsterdam, 1975

sa N. Santoro, Efficient abstract Implementations For Relational Data Structures, Ph.D. Thesis, 1980, University of Waterloo

To F. W. Tompa, Data Structure Design, in data structures computer Graphs and Pattern Recognition, Academic Press, New York, 1978

TSU T. Tsuji, J. Toyoda, K. Tanaka, relational data graphs and some properties of them, J. Comp. system sci. 15 (1977)

Ull Jeffry D. Ullman, Principles of data base systems, Computer science Press, 1980

Appendix A

A SOFTWARE PACKAGE FOR EFFICIENT ABSTRACT STRUCTURES

A.1 DOCUMETATION

The implementation of the theoretical results presented throughout this thesis can be viewed as two main procedures: Procedure SPACOPT and porcedure TIMIMP.

Procedure SPACOPT (for space optimization) reads in the number of relations of the model and the constraints. The constraints are read into a matrix. Each row of the matrix corresponds to a constraint.. The first entry of each row indicates the number of elements of the constraint including that entry. The second entry, is the left hand side of the equality and the remaining entries are the names of the re-

lations that sould be joined to give the relation of the left hand side. For example, if we have a constraint of the form:

7 = 5*3*1

the first five entries of the corresponding row are: 5,7,5,3 and 1.

The objective of the procedure SPACOPT is to determine the two components of the minimal perfect implementation of the model, which are the set of explicitly stred relations and the query mapping. To achieve that it creates a matrix MATQUERYMAP similar to the one corresponding to the constraints. We will refer to the query mapping by MATQUERYMAP and to the set of constraints by MATX.

Procedure SPACOPT initializes the first row of MATQUERYMAP to the first row of MATX. Then repeatedly, until MATX contains no more constraints checks for each row whether it could be added to the MATQUERYMAP, and if yes it will add it after replacing any occurrence in the MATQUERYMAP of its second entry by its equivalent which is the rest of the row, then delete it from the MATX and delete any row of MATX

if it has same second entry or, if replacing an entry of that row which is equal to the second entry of this row will create a cycle. A row cannot be added to MATQUERYMAP iff replacing one occurrence of its second entry in the rows of MATQUERYMAP will create a cycle. After all the rows of MATX have been processed, MATQUERYMAP is formed. Then the set of explicitly stored relations is computed by checking all second entries of MATQUERYMAP. All relations which figures as second entry of MATQUERYMAP are not explicitly stored. The rest forms the set of explicitly stored relations. Procedure SHOWOUTP is then called to display the elements of MATQUERYMAP and the set of explicitly stored relations.

After determining the components of the minimum perfect implementation, procedure SPACOPT calls procedure TIMIMP. The TIMIMP procedure is an implementation of the redundant graph. SPACOPT passes its output to procedure TIMIMP, which asks for more informations. TIMIMP represents the graph representing the domains and the binary relations among them by a boolean matrix. For this end the domains are numbered from 1 to N, where N is the number of domains. Each domain number corresponds to a row number and a column number of

the matrix. The entries of the matrix are defined as follows:

a(i,j) = 1 if a relation between i and j exists

a(i,j) = 0 otherwise

In order for TIMIMP to form the boolean matrix it needs to know for each relation what are the domains related by it. Since the relations are also numbered from 1 to NRELATIONS then a matrix of two columns and NRELATIONS rows is used. SETREL is the array used to represent these informations. If i is a relation between domains m and n then SETREL(i,1)=m and SETREL(i,2)=n. After reading these informations TIMIMP calls a sequence of procedures which themselves calls other procedures. The goal of all these procedures is to form the redundant graph by executing the algorithm of page 50.

TIMIMP calls REPRES to convert the output of SPACOPT into a boolean matrix representing the graph of the minimum perfect implementation. The graph is a tree. Then calls ADDREL.

ADDREL is a recursive procedure. It takes a tree and its root and calls BESTPART to determine the node with best partition of this tree then calls CONNECT to connect this node to all its non-direct child, then determines the names of the relation between this best partition node

A.2 THE PROGRAM

```
PROGRAM PROCESSING(INPUT,OUTPUT);

  (* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   *                                                                     *
   *                    PROCEDURE  PROCESSING                            *
   *                    ---------------------                            *
   *                                                                     *
   *      PROCEDURE PROCESSING READS A SET OF RELATIONS                  *
   *      AND A SET OF CONSTAINTS. FOR EACH CONSTRAINT                   *
   *      IT DETERMINES WHETHER A RELATION CAN COMPUTED                  *
   *      THROUGH OTHER RELATIONS WITHOUT CREATING A CYCLE               *
   *      BY COMPARING IT TO THE QUERY MAPPING AND IF THE                *
   *      ANSWER IS POSITIF THAN IT WILL ADD THAT CONSTRAINT             *
   *      TO THE QUERY MAPPING AFTER REPLACING EACH OCCURENCE            *
   *      OF THAT RELATION BY ITS EQUIVALENT IN THE QUERY                *
   *      MAPPING AND IN THE REST OF THE CONSTRAINTS. AS                 *
   *      A RESULT THIS PROCEDURE GIVES THE SET OF                       *
   *      EXPLICITLY STORED RELATIONS AND THE QUERY                      *
   *      MAPPING OF THE MINIMAL PERFECT IMPLEMENTATION.                 *
   *                                                                     *
```

```
   * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*)

TYPE MATRIX =  ARRAY(.1..50,1..20.)  OF INTEGER ;   VECTOR =
ARRAY(.1..50.)       OF INTEGER ; VAR

  MATX,MATQUERYMAP                                                    :
MATRIX ;

  ELEMTOSUBS ,SUBSTINCONS, SUBSINCONSTOP, REL,ORIGREL                 :
VECTOR ;
  NRELATIONS, NCONSTRAINTS, REMNUMOFCONS,I, J, FLAG,
  JJ, NORDREL,  NORRFOLLREL, SUBSTITNUM, ELEMTOSBS,
  NNRELATIONS,NUMOFSUBSTIN,  NNCONSTRAINTS, NQ, NQUERY,
  NFIRSTMEMBER, NDELETEDCONS, P, PP,NORDREL2,

  NELEQUERYMAP, NN, FLAG2, POSSUBST, LL, L, NEXREL                    :
INTEGER ;

  (* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   *                                                                    *
   *            PROCEDURE  SEXREL                                       *
   *            ----------------                                        *
   *                                                                    *
```

```
 *    THIS PROCEDURE WILL TAKE THE SET OF ORIGINAL          *
 *    RELATIONS OF THE MODEL, AS INPUT AND SEARCH           *
 *    THE QUERY MAPPING FOR THE RELATIONS THAT CAN          *
 *    BE DERIVED THROUGH COMPUTATIONS AND PRODUCE AS        *
 *    OUTPUT THE SET OF RELATIONS THAT NEED TO BE           *
 *    EXLICITLY STORED.                                     *
 *                                                          *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)

PROCEDURE SEXREL ;

VAR
   I, J, FLAG : INTEGER ;

BEGIN
FOR I := 1 TO NRELATIONS  DO
 BEGIN
   FLAG := 1 ;
   FOR J := 1 TO NELEQUERYMAP DO
   IF (ORIGREL(.I.) = MATQUERYMAP(.J,2.)) THEN
   FLAG := 0 ;
   IF FLAG = 1 THEN
   BEGIN
      NEXREL := NEXREL + 1 ;
```

```
      REL(.NEXREL.) := ORIGREL(.I.)
      END
   END
END ;

(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                                                                     *
 *              PROCEDURE  SHOWOUTP                                    *
 *              ------------------                                     *
 *                                                                     *
 *     THIS PROCEDURE PRINT TWO CONPONENTS OF THE                      *
 *     ABSTRACT STRUCTURE AS PRODUCED BY THE OTHER                     *
 *     PROCEDURES. THE TWO COMPONENTS ARE THE SET                      *
 *     OF EXPLICITLY STORED RELATIONS AND THE QUERY                    *
 *     MAPPING. IT PRINT THE NUMBER OF EXPLICITLY                      *
 *     STORED RELATIONS FIRST AND THEN THEIR NAMES                     *
 *     AND THEN THE NUMBER OF ELEMENTS IN THE QUERY                    *
 *     MAPPING AND THE QUERY MAPPING.                                  *
 *                                                                     *
 * * * * * * *  * * * * * * * * * * * * * * * * * * * * * * * * * * *
*)
```

```
PROCEDURE SHOWOUTP ;

VAR
  I, J  : INTEGER ;

BEGIN
  WRITELN ;
  WRITELN('  THERE ARE ',NEXREL :  3,'  EXPLICITLY  STORED
RELATIONS') ;
  WRITELN ;
  WRITELN(' THESE ARE ') ;
  WRITELN ;
  FOR J:= 1 TO (NEXREL DIV 10) + 1  DO
  BEGIN
    FOR I := 1 TO 10 DO
    IF (I+(J-1)*10) <= NEXREL THEN
    WRITE (' R', REL(.I+ (J-1)*10.) : 2,',') ;
    WRITELN
    END ;
  WRITELN ;
  WRITELN ;
  WRITE(' THE QUERY MAPPING IS' ) ;
  WRITELN ;
```

```
FOR I := 1 TO NELEQUERYMAP DO
BEGIN
  WRITELN ;
  WRITE(' R', MATQUERYMAP(.I,2.) : 2,' = ') ;
  FOR J := 3 TO MATQUERYMAP(.I,1.) DO
  BEGIN
    WRITE(' R', MATQUERYMAP(.I,J.) : 2 ) ;
    IF J < MATQUERYMAP(.I,1.) THEN
    WRITE('*')
    END
  END
END ;


(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                                                               *
 *          PROCEDURE  TIMIMP                                    *
 *          ----------------                                     *
 *                                                               *
 * THIS PROCEDURE TAKES THE SET OF RELATIONS DETERMINED          *
 * BY THE MINIMAL PERFECT IMPLEMENTATION AS ITS INPUT            *
 * AND ADD SOME MORE RELATIONS ACCORDING TO SOME SCHEMA          *
 * IN ORDER TO REDUCE THE TIME REQUIRED TO ANSWER A              *
```

```
    * QUERY.                                                    *
    *                                                           *
    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*)

PROCEDURE TIMIMP  ;

VAR
    R,NODEBESTPART, BSTPARTI,N,J,I      : INTEGER ;
    SETREL                              : ARRAY(.1..90,1..2.)
OF INTEGER ;
    MAT, MATREL                         : MATRIX ;

(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
    *                                                           *
    *                  PROCEDURE  COUNT                         *
    *                  ---------------                          *
    * THIS PROCEDURE COUNT TAKES A SUBTRE AND ITS ROOT AS       *
    * INPUT, EXECUTE A DEPTH FIRST SEARCH AND DETERMINE         *
    * THE NUMBER OF NODES IN THIS SUBTREE.                      *
    *                                                           *
```

```
   * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

*)


PROCEDURE COUNT(MATX      : MATRIX ;
                M,CHILD,N : INTEGER;
                VAR WEI   : INTEGER );

VAR
   I  : INTEGER ;

BEGIN
  WEI := WEI + 1 ;
  FOR I := 1 TO N DO
  IF (MAT(.CHILD,I.) =1 ) AND ( I<>M ) THEN
  COUNT (MAT,CHILD,I,N,WEI)
  END;


  (* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   *                                                                   *
   *              PROCEDURE  PART                                      *
   *              --------------                                       *
   *                                                                   *
```

```
    *     THIS PROCEDURE TAKES A SUBTREE AND ITS ROOT AS      *
    *     ITS INPUT AND DETERMINES THE PARTITION OF THE       *
    *     NODES IN THAT SUBTREE.                              *
    *                                                         *
    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*)


  PROCEDURE PART(MATX          : MATRIX    ;
                 N, ROOT       : INTEGER   ;
                 VAR PARTI     : INTEGER ) ;

  VAR
    J, MAXWEI, MINWEI, SUM,
     NBRNCH, WEIGHT            : INTEGER   ;

  BEGIN
    MAXWEI := 0 ;
    MINWEI := N + 1 ;
    SUM := 0 ;
    PARTI := 0 ;
    NBRNCH := 0 ;
    FOR J := 1 TO N DO
```

```
IF MATX(.ROOT,J.) = 1 THEN
BEGIN
  NBRNCH := NBRNCH + 1 ;
  WEIGHT := 0 ;
  COUNT(MATX,ROOT,J,N,WEIGHT) ;
  IF WEIGHT > MAXWEI THEN
  MAXWEI := WEIGHT ;
  IF WEIGHT < MINWEI THEN
  MINWEI := WEIGHT ;
  SUM := SUM + WEIGHT
  END ;
IF NBRNCH  = 1   THEN
PARTI := WEIGHT
 ELSE
 IF (MAXWEI - MINWEI ) < ABS(SUM - 2*MAXWEI) THEN
 PARTI := MAXWEI-MINWEI
 ELSE
 PARTI := ABS(SUM-2*MAXWEI)
 END ;


(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                                                                   *
```

```
(*  *********************************************************
    *                                                       *
    *                PROCEDURE  BESTPART                    *
    *                ___________________                    *
    *                                                       *
    *     THIS PROCEDURE TAKES A TREE AS ITS INPUT AND      *
    *     SEARCHES ALL ITS NODES TO DETERMINE THE ONE       *
    *     WHICH HAS THE BEST PARTITION. A NODE IS THE       *
    *     PARTITION NODE IF THE DIFFERENCE IN THE           *
    *     NUMBER OF NODES OF THE SUBTREES ROOTED AT         *
    *     THAT NODE IS MINIMUM.                             *
    *                                                       *
    *********************************************************
*)

PROCEDURE BESTPART(MATX                 : MATRIX      ;
                   ROOT, N, FUTHER      : INTEGER     ;
                   VAR NBP, BSTPARTI    : INTEGER  )  ;

VAR
  J, PARTI                              : INTEGER     ;

BEGIN
  PART(MATX,N,ROOT,PARTI) ;
  IF PARTI < BSTPARTI THEN
```

```
    BEGIN
    BSTPARTI := PARTI ;
    NBP := ROOT
    END ;
    FOR J := 1 TO N DO
      IF ( MATX(.ROOT,J.) = 1 ) AND (J <> FUTHER ) THEN
        BESTPART(MATX,J,N,ROOT,NBP,BSTPARTI)
END ;

(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                                                               *
 *                PROCEDURE  CONNECT                             *
 *                ------------------                             *
 *                                                               *
 *     THIS PROCEDURE TAKES A TREE, A NODE AND ITS               *
 *     FATHER AS ITS INPUT, AND CONNECT IT TO ALL                *
 *     THE NODES IN THE SUBTREE FOR WHICH ITS A ROOT             *
 *     EXCEPT THE ONES WHICH ARE ITS DIRECT CHILD.               *
 *                                                               *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*)
```

```
PROCEDURE CONNECT (MATX                          : MATRIX    ;
                   FATHER,NODEBESTPARTI,ROOT     : INTEGER   ;
                   VAR MATREL                    : MATRIX  ) ;

VAR
  J                                              : INTEGER   ;

BEGIN FOR J:= 1 TO N DO
  IF (MATX(.ROOT,J.) = 1) AND (J <> FATHER ) THEN
  BEGIN
    MATREL(.NODEBESTPARTI,J.) := 1 ;
    CONNECT(MATX,ROOT,NODEBESTPARTI,J,MATREL)
    END END ;


  (* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   *                                                               *
   *                  PROCEDURE  CHECK                             *
   *                  ---------------                              *
   *                                                               *
   *     THE PROCEDURE CHECK TAKES 2 DOMAINS D1 AND D2 AS          *
   *     INPUTS, AND DETERMINE WHETHER A RELATION BETWEEN          *
   *     THESE TWO DOMAINS EXISTS AND IF YES DETERMINE             *
```

```
   *     THAT RELATION OTHERWISE IT WILL ADD IT TO THE      *
   *     TO THE ORIGINAL SET OF RELATIONS.                  *
   *                                                        *
   * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*)

PROCEDURE CHECK(D1,D2        : INTEGER   ;
                VAR RELTION : INTEGER ) ;

VAR
  I                          : INTEGER   ;

BEGIN
  I := 1;
  RELTION := 0 ;
  WHILE ( I <= NRELATIONS ) AND (RELTION = 0 ) DO
  BEGIN
    IF (SETREL (.I,1.) = D1 ) AND (SETREL(.I,2.) = D2 ) THEN
    RELTION := I
    ELSE
    IF (SETREL(.I,1.) = D2 ) AND (SETREL(.I,2.) = D1 ) THEN
    RELTION := -I ;
    I := I + 1
```

```
  END ; IF(RELTION = 0 ) THEN BEGIN
NRELATIONS:= NRELATIONS + 1 ;
SETREL(.NRELATIONS,1.) := D1 ;
SETREL(.NRELATIONS,2.) := D2 ;
RELTION := NRELATIONS
END END ;
```

```
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *                                                                     *
 *                  PROCEDURE  UPJOIN                                  *
 *                  ----------------                                   *
 *                                                                     *
 *     PROCEDURE UPJOIN WORKS RECURSIVLY. IT TAKES A                   *
 *     NODE WITH OF BEST PARTITION OF A TREE AND AND                   *
 *     CONNECT TO ALL THE NODES IN THE TREE BUT THE                    *
 *     ONES WHICH ARE NOT ITS DIRECT CHILD. THEN                       *
 *     IT CALLS THE 'UPDAQM' PROCEDURE TO UPDATE                       *
 *     THE QUERY MAPPING.                                              *
 *                                                                     *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*)
```

```
PROCEDURE UPJOIN(ROOT,FATHER,NODEBESTPART  : INTEGER   ;
                 VAR RELAT                 : INTEGER ) ;

VAR

  J, NEXTREL, COMPREL                      : INTEGER   ;

  (* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   *                                                               *
   *                  PROCEDURE  UPDAQM                            *
   *                  ----------------                             *
   *      PROCEDURE UPDAQM TAKES A RELATION AND CHECKS             *
   *      WHETHER IT IS ALREADY IN THE QUERY MAPPING               *
   *      AND IF IT IS NOT THEN WILL ADD IT TO IT.                 *
   *                                                               *
   * * * * * * *  * * * * * * * * * * * * * * * *  * * * * * * * *
*)

PROCEDURE UPDAQM ;

VAR
```

```
  I, J, J1, T1, FLAG                    :  INTEGER   ;
BEGIN
  T1 := 0 ;
  I := 1 ;
  FLAG := 1 ;
  WHILE (I <= NELEQUERYMAP ) AND (FLAG = 1 ) DO
  BEGIN
    IF (MATQUERYMAP(.I,2.) = COMPREL ) OR
    (MATQUERYMAP(.I,2.) = - COMPREL) THEN
    BEGIN
      FOR J:= I TO NELEQUERYMAP - 1 DO
      FOR J1 := 1 TO MATQUERYMAP(.J+1,1.) DO
        MATQUERYMAP(.J,J1.) := MATQUERYMAP(.J+1,J1.) ;
      FLAG := 0;
      NELEQUERYMAP := NELEQUERYMAP - 1;
      NEXREL := NEXREL + 1 ;
      REL(.NEXREL.) := COMPREL
      END ;
    I := I + 1
  END ;
  FOR I := 1 TO NELEQUERYMAP DO
  BEGIN
```

```
J := 3 ;
T1 := 0 ;
FLAG := 1 ;
WHILE ( J <= MATQUERYMAP(.I,1.) - 1 ) AND (FLAG = 1 ) DO
BEGIN
  IF (MATQUERYMAP(.I,J.) = RELAT    ) AND
  (MATQUERYMAP(.I,J+1.) = NEXTREL) THEN
  BEGIN
    T1 := MATQUERYMAP(.I,J.) ;
    MATQUERYMAP(.I,J.) := COMPREL
    END
  ELSE
  IF(MATQUERYMAP(.I,J.) = -NEXTREL) AND
  (MATQUERYMAP(.I,J+1.) = -RELAT )  THEN
  BEGIN
    T1 := MATQUERYMAP(.I,J.) ;
    MATQUERYMAP(.I,J.) := -COMPREL
    END ;
  IF (( T1 = RELAT) AND
  (MATQUERYMAP(.I,J+1.)=NEXTREL))
  OR ((T1 = -NEXTREL)
  AND (MATQUERYMAP(.I,J+1.)= -RELAT)) THEN
```

```
    BEGIN
      IF(MATQUERYMAP(.I,1.) > J ) THEN
      FOR J1 := J + 1 TO MATQUERYMAP(.I,1.) - 1 DO
      MATQUERYMAP(.I,J1.) := MATQUERYMAP(.I,J1 + 1 .) ;
      MATQUERYMAP(.I,1.) := MATQUERYMAP(.I,1.) - 1  ;
      FLAG := 0
      END ;
    J := J + 1
    END
  END
END ;
BEGIN
  FOR J := 1 TO N DO
  IF (MAP(.ROOT,J.) = 1 ) AND (J <> FATHER) THEN
  BEGIN
    CHECK(ROOT,J,NEXTREL) ;
    CHECK(NODEBESTPART,J,COMPREL) ;
    UPDAQM   ;
    UPJOIN(J,ROOT,NODEBESTPART,COMPREL)
    END
  END ;
```

```
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                                     *
*                    PROCEDURE  ADDREL                                *
*                    ----------------                                 *
*     PROCEDURE ADDREL TAKES A TREE AS ITS INPUT                      *
*     DETERMINES RECURSIVLY THE NODE WITH BEST                        *
*     PARTITION AND CONNECT TO ALL THE NODES OF THE                   *
*     TREE ROOTED AT THAT NODE. THEN IT ADDS THE                      *
*     THE CORRESPONDING RELATION TO THE SET OF                        *
*     EXPLICITLY STORED RELATIONS AFTER IT UPDATE                     *
*     THE QUERY MAPPING.                                              *
*                                                                     *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*)

PROCEDURE ADDREL(R,N                    :INTEGER   ;
                 VAR MATX,MATREL        : MATRIX ) ;

VAR
  J,NBSP,NODEBESTPARTI, REL             : INTEGER  ;

BEGIN
```

```
  NBSP := N+1 ;
  BESTPART(MATX,R,N,N+1,NODEBESTPARTI,NBSP) ;
  CONNECT(MATX,N+1,NODEBESTPARTI,NODEBESTPARTI,MATREL) ;
  FOR J := 1 TO N DO
  IF MATX(.NODEBESTPARTI,J.) = 1 THEN
  BEGIN
    CHECK(NODEBESTPART,J,REL) ;
    UPJOIN(J,NODEBESTPARTI,NODEBESTPARTI,REL) ;
    MATX(.NODEBESTPARTI,J.) := 0 ;
    MATX(.J,NODEBESTPARTI.) := 0 ;
    ADDREL(J,N,MATX,MATREL)
    END END ;


PROCEDURE REPRES ;

  (* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
   *                                                               *
   *                    PROCEDURE  REPRES                          *
   *                    ----------------                           *
   *                                                               *
   *      PROCEDURE REPRES TAKES A SET OF DOMAINS AND A SET        *
```

```
    *      OF RELATIONS AMONG THEM AND REPRESENT THEM AS      *
    *      A BOOLEAN MATRIX.                                  *
    *                                                         *
    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*)

VAR
  I , J : INTEGER ;

BEGIN
  FOR I := 1 TO N DO
  FOR J := 1 TO N DO
  BEGIN
    MAT(.I,J.) := 0 ;
    MAT(.J,I.) := 0
    END ;
  FOR I := 1 TO N -1 DO
  BEGIN
    MAT(.SETREL(.REL(.I.),1.),SETREL(.REL(.I.),2.).) := 1 ;
    MAT(.SETREL(.REL(.I.),2.),SETREL(.REL(.I.),1.).) := 1
    END;
  END ; (* END PROCEDURE REPRES *)
```

```
BEGIN
  FOR I:= 1 TO NRELATIONS DO
  READ(SETREL(.I,1.),SETREL(.I,2.) ) ;
  READ ( N ) ;
  REPRES ;
  ADDREL(1,N,MAT,MATREL) ;
  WRITELN ;
  WRITELN ;
  WRITELN ;
  WRITELN ;
  WRITELN ;
  WRITELN ;
  WRITELN ;
  WRITELN ;
  WRITELN ;
  WRITE('   THE   ABSTRACT   STRUCTURE   CORRESPONDING   TO
REDUNDANT GRAPH IS');
  WRITELN ;
  SHOWOUTP ;
 END ;
 BEGIN
  READ(NNRELATIONS,NNCONSTRAINTS);
```

```
  NRELATIONS:=NNRELATIONS;
  NCONSTRAINTS:=NNCONSTRAINTS;
(*  READ THE CONSTRAINTS *)
  FOR I :=1 TO NCONSTRAINTS DO
  BEGIN
    READ (MATX(.I,1.)) ;
    FOR J:=2 TO MATX(.I,1.) DO
    READ (MATX(.I,J.)) ;
    END;
  I:=1;
  REMNUMOFCONS:=NCONSTRAINTS;
  NELEQUERYMAP := 0;
  WHILE REMNUMOFCONS >= 1 DO
(* FIND OUT IF POSSIBLE TO SUBSTITUTE FOR I *)
  BEGIN
    J:=1;
    SUBSTITNUM:=0; FLAG:=1;
    IF NELEQUERYMAP <> 0 THEN
    WHILE (FLAG <> 0) AND (J <= NELEQUERYMAP) DO
    BEGIN
      JJ:=3;
      WHILE (FLAG<>0) AND (JJ <= MATQUERYMAP(.J,1.)) DO
```

```
        BEGIN
          IF (MATX(.I,2.) =  MATQUERYMAP(.J,JJ.)) OR
            (MATX(.I,2.) = -MATQUERYMAP(.J,JJ.)) THEN
          BEGIN
            NORRFOLLREL:=3;
            WHILE (FLAG<>0) AND (NORRFOLLREL <= MATX(.I,1.))
DO
            BEGIN
              NORDREL:=3;
              WHILE   (FLAG   <>   0)   AND   (NORDREL   <=
MATQUERYMAP(.J,1.))  DO
              IF       MATX       (.I,NORRFOLLREL.)       =
MATQUERYMAP(.J,NORDREL.) THEN
                FLAG:=0
              ELSE
                NORDREL:=NORDREL+1;
              NORRFOLLREL:=NORRFOLLREL+1
              END;
              IF FLAG=1 THEN
              BEGIN
                SUBSTITNUM:=SUBSTITNUM+1;
                SUBSTINCONS(.SUBSTITNUM.) := J;
```

```
                ELEMTOSUBS(.SUBSTINCONS(.  SUBSTITNUM.).) :=
JJ
                 END
               END;
               JJ:=JJ+1
             END ;
           J := J+1
         END ;
         IF FLAG = 1 THEN
         BEGIN
           NELEQUERYMAP := NELEQUERYMAP + 1 ;
           FOR P := 1 TO MATX(.I,1.) DO
           MATQUERYMAP(.NELEQUERYMAP,P.) := MATX(.I,P.) ;
        (* SUBSTITUTE IF POSSIBLE IN THE REST OF THE
CONSTRAINTS
            ELSE DELETE THAT CONSTRAINT IF CONFLICT *)
           POSSUBST := REMNUMOFCONS - 1 ;
           NN := I + 1;
           FLAG2 := 1 ;
           WHILE (POSSUBST >= 1) DO
           BEGIN
           IF (MATX(.I,2.) <> MATX (.NN,2.)) AND
```

```
            (MATX(.I,2.) <>'-MATX(.NN,2.)) THEN
            BEGIN
              NORRFOLLREL := 3 ;
              WHILE   (FLAG2<>0)    AND    (NORRFOLLREL   <=
MATX(.NN,1.) ) DO
              BEGIN
               IF (MATX(.I,2.) = MATX(.NN,NORRFOLLREL.)) OR
                  (MATX(.I,2.)   =  -MATX(.NN,NORRFOLLREL.))
THEN
                 BEGIN
                   NORDREL := 3 ;
                   WHILE (FLAG2 <> 0)  AND (NORDREL <= MATX
(.I,1.)) DO
                   BEGIN
                     NORDREL2 := 3;
                     WHILE (FLAG2 <> 0) AND
                           (NORDREL2 <= MATX(.NN,1.)) DO
                     BEGIN
                     IF        (MATX(.I,NORDREL.)        =
MATX(.NN,NORDREL2.)) OR
```

```
                    (MATX(.I,NORDREL.)                    =
-MATX(.NN,NORDREL2.)) THEN
                   BEGIN
                     REMNUMOFCONS := REMNUMOFCONS - 1;
                     IF NCONSTRAINTS > NN THEN
                       FOR P  := NN  TO NCONSTRAINTS  - 1
DO
                        FOR PP := 1 TO MATX(.P+1,1.) DO
                         MATX(.P,PP.) := MATX(.P+1,PP.);
                     NCONSTRAINTS := NCONSTRAINTS - 1;
                     FLAG2 := 0;
                     NORRFOLLREL := MATX(.NN,1.)
                   END;
                   NORDREL2 := NORDREL2 + 1
                 END;
                 NORDREL := NORDREL + 1
               END;
             IF FLAG2 = 1 THEN
             BEGIN
               LL := MATX(.NN,1.) - NORRFOLLREL ;
               IF LL <> 0 THEN
```

```
                    FOR P := 1 TO MATX(.NN,1.)  - NORRFOLLREL
DO
                     MATX(.NN,NORRFOLLREL+MATX(.I,1.)-3  +  P
.)
                        :=MATX(.NN,NORRFOLLREL+P.);
                    IF  MATX(.I,2,)  =  MATX(.NN,NORRFOLLREL.)
THEN
                     FOR P := 3 TO MATX(.I,1.)  DO
                      MATX(.NN,NORRFOLLREL+P-3.)          :=
MATX(.I,P.)
                    ELSE
                     FOR P:= 3 TO MATX(.I,1.) DO
                      MATX(.NN,NORRFOLLREL+P-3.)
                                          :=
MATX(.I,MATX(.I,1.)-P+3.);
                    MATX(.NN,1.) := MATX(.NN,1.) + MATX(.I,1.)
-3 ;
                     NORRFOLLREL := MATX(.NN,1.)
                     END
                  END ;
                  NORRFOLLREL := NORRFOLLREL + 1
                  END ;
```

```
                POSSUBST := POSSUBST - 1;
                NN:=NN +1
                END
                ELSE
(* DELETE CONSTRAINT OF SAME FIRST MEMBER  *)
                BEGIN
                  FOR P := NN TO NCONSTRAINTS - 1 DO
                   FOR PP := 1 TO MATX(.P+1,1.) DO
                    MATX(.P,PP.) := MATX(.P+1,PP.);
                  NCONSTRAINTS := NCONSTRAINTS - 1;
                  REMNUMOFCONS := REMNUMOFCONS - 1;
                  POSSUBST := POSSUBST - 1
                  END
                END;
(* SUBSTITUTION IN THE QUERY MAPPING   *)
                IF SUBSTITNUM <> 0 THEN
                 FOR NQ := 1 TO SUBSTITNUM DO
                 BEGIN
                   LL :=  MATQUERYMAP(.SUBSTINCONS(.NQ.),1.)
```

```
                                        ELEMTOSUBS(.SUBSTINCONS(.NQ.).);

                    IF LL > 0 THEN
                     FOR L := 1 TO LL DO
                      MATQUERYMAP(.SUBSTINCONS(.NQ.),
                      ELEMTOSUBS(.SUBSTINCONS(.NQ.).)        +
MATX(.I,1.)+L-3.)
                          :=MATQUERYMAP(.SUBSTINCONS(.NQ.),
                            ELEMTOSUBS(.SUBSTINCONS(.NQ.).) +
L.) ;
                    IF              MATX(.I,2.)              =
MATQUERYMAP(.SUBSTINCONS(.NQ.),
                     ELEMTOSUBS(.SUBSTINCONS(.NQ.).).) THEN
                     FOR P:= 3 TO MATX(.I,1.) DO
                      MATQUERYMAP(.SUBSTINCONS(.NQ.),ELEMTOSUBS
                       (.SUBSTINCONS(.NQ.).)  + P -  3 .)  :=
MATX(.I,P.)
                    ELSE
                     FOR P := 3 TO MATX(.I,1.) DO
                       MATQUERYMAP(.SUBSTINCONS(.NQ.),
```

```
                        ELEMTOSUBS(.SUBSTINCONS(.NQ.).)+
P - 3 .)
                          :=   MATX(.I,MATX(.I,1.) - P +
3 .) ;
                    MATQUERYMAP(.SUBSTINCONS(.NQ.),1.)
                          :=
MATQUERYMAP(.SUBSTINCONS(.NQ.),1.)
                              + MATX(.I,1.) - 3
                END
            END ;

            I := I + 1 ;
            REMNUMOFCONS := REMNUMOFCONS - 1;
        END;
        FOR I := 1 TO NRELATIONS DO
         BEGIN
           ORIGREL(.I.) := I ;
           END ;
           NEXREL := 0;
           SEXREL ;
           WRITELN ;
           WRITELN ;
           WRITELN ;
```

```
                WRITELN ;
                WRITELN ;
                WRITELN ;
                WRITE(' THE MINIMAL  PERFECT IMPLEMENTATION
CORRESPOND');
                WRITE(' TO  THE FOLLOWING SET  OF RELATIONS
AND QUERY');
                WRITE('MAPPING');
                WRITELN ;
                SHOWOUTP ;
                TIMIMP ;
              END.
```