A MEMBERSHIP ALGORITHM FOR FUNCTIONAL DEPENDENCIES IN RELATIONAL DATABASE

Dissertation
submitted to Jawaharlal Nehru University
in partial fulfilment of the requirements for the award of
the Degree of
MASTER OF PHILOSOPHY

RAMA CHARAN TRIPATHY

SCHOOL OF COMPUTER AND SYSTEM SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI
1986

DECLARATION

The research work entitled "A Membership Algorithm for Functional Dependencies in Relational Database" embodied in this dissertation has been carried out at the School of Computer and System Sciences, Jawaharlal Nehru University, New Delhi. This work is original and has not been submitted so far in part or full to any other University or institution for the award of any other degree or diploma.

R. C. Tripathy 12.6.86

12 JUN 1986

Rama Charan Tripathy

Prof. K. K. Nambiar

Dean

Dr. P. C. Saxena

Supervisor

Dedicated to

my respected father,

from whom I learned

patience and perseverance.

ACKNOWLEDGEMENT

I wish to express my sincere appreciation and deep gratitude to my supervisor, Dr. P. C. Saxena, Associate Professor, School of Computer and System Sciences, Jawaharlal Nehru University, New Delhi for his keen interest, capable guidance, invaluable encouragement, inspiration for hard work, constructive criticisms, advice, patience and wisdom during my M.Phil coursework and particularly throughout the planning and completion of this dissertation.

I am indebted to Professor K. K. Nambiar, Dean, School of computer and System Sciences, Jawaharlal Nehru University for providing me all kinds of facilities and co-operation in completing this work and with whom I have the pleasure of having free discussions.

My thanks are also to the faculty members and other staff of the school for providing me help and guidance.

In working so long on my M.Phil degree, I have been sustained by the relationships and help of many friends. My thanks to Mr.P. S. Acharya and Mr. Surendra Kumar for their co-operation and helpful discussions.

My special gratitude to my family for their continuous encouragement and support throughout my academic career.

Finally I am grateful to Jawaharlal Nehru University, New Delhi for providing me the financial assistance.

Rama Charan Tripathy

TABLE OF CONTENTS

PREFACE

One of the most significant contributions to the Data Base Management Technology in recent years has been the development of relational point of view of a data base. The relational model of data base formalizes the organization of and access to highly structured data. The model provides a view of data that is elegant in its simplicity and encourages the application of abstract mathematical reasoning. A substantial amount of research activity has surrounded the field since its inception in the work of Codd in 1969. In that time, several issues related to the model have been studied intensively. Among them are :

1. The characterization of semantic constraints on the data, and
2. The selection of data base design scheme.

The theoretical work in this area will be of little utility without a major effort at reducing theory to practice. Thus, there is a need for the design and analysis of algorithms that are required for a good data base system.

Let us first describe the most important area of relational data base which is the issue of semantic constraints and may be given in various disguise. For example, values may be constrained to lie with in certain domains, as in "salaries may not be more then Rs.10,000. "Such constraints take the form of predicates ranging over single domains of salaries. Another class of constraints take the form of "values from domain X depends on values from domain Y according to rule Z ".The classic example of data dependency is the functional dependency, which asserts that, values for one set of attributes are uniquely determined by another set of attributes. An example of functional dependency is the statement:

> " No employee of a company has two different salaries."

Functional dependencies were first described in full generality by Codd in 1971, although they have been recognized for a long time taking the form of keys. Many other forms of data dependencies have been described, e.g. multivalued dependencies, join dependencies, first-order hierarchical dependencies and implicational dependencies.

In comparison to some other varieties of dependencies, functional dependencies are easily understandable and readily identifiable. Furthermore functional dependencies are special cases of nearly all other types of dependencies. A comprehensive understanding of functional dependencies can help to formulate the practical implication of other types of dependencies. All the work on functional dependencies are based on the axiomatization of functional dependencies developed by Armstrong in 1974. Wang and Wedekind proposed an algorithm to synthesize the relational schema from a set of functional dependencies. Another approach was made by Bernstein to design a relational data base schema from a given set of functional dependencies. Every problem dealing with functional dependencies requires a manipulation of functional dependencies according to the axioms put forth by Armstrong. In 1979 Bernstein has given a graph theoretic approach - a tree model for the derivation of functional dependencies and also proposed a procedure to solve the membership problem for functional dependencies.

The main objective of the approach given in this dissertation is also to solve the membership problem for functional dependencies. It has shown here that, the proposed algorithm is faster than the algorithm proposed by Bernstein.

The organization of the dissertation is as follows: In first chapter the basic concepts of the relational model for databases are given and also a detailed discussion about data dependencies, normalization and normal forms are presented.

In chapter two the theoretical bases for the data dependencies in general, and the implication problem of functional dependencies, multivalued dependencies and join dependencies in particular, are discussed.

The chapter three starts with a simple algorithm for the membership problem for functional dependencies and then the algorithm has been modified by using a simple data structure for implementation purpose.

I. INTRODUCTION

1.1 Data Base Management Systems

Data Base System was introduced in late 1960's to overcome the problems arising in the use of conventional file systems, such as: (1) The inability to efficiently integrate numerous large files and (2) The inability to support higher level data and file organizations. The major elements of a data base system are, the data base, queries and their query programmes, file organization and data management functions. The data base is the repository of all data of interest to the user of the system. The queries in their form in the computer as query programmes, represent the users in the system and create its actions. File organization is necessary to expedite operations, and the data management functions represent the set of all operative programmes in the data base system, necessary for storage retrieval and space management.

There are three approaches to define the data base. The first approach is from designer's point of view and deals with the technical aspects of data base technology, and is defined by Cardenas [11] as " a database is a collection of occurrences of record types, where the record types are interrelated by means of specific relationships".

The second approach is concerned with its application point of view from the standpoint of organization. According to Mandell, " a data base is a grouping of data elements, structured to fit the information need of an organization".

Date ([15],1981,p-7) states, "a database is a collection of occurrences of stored operational data used by the application systems of some particular enterprise".

The third approach is a combination of both technical point of view and application view of the database technology, and is defined by Kroenke as " a data base allows an organization's data to be processed as an integrated whole. It reduces artificiality imposed by separate files for separate applications and permit users to access data in a manner which is more natural to them".

A Data Base Management System (DBMS) is a software interface between the user and the physical storage of data in the computer, which handles the physical storage and the retrieval of information in a database. It also has the important capability of associating any logical relationships of data elements (for example, employee names are logically associated with Job-skill classifications, Salary, Department, etc.), regardless of the physical organization or location of the individual data elements in the database. Hence all the different logical relationships of data elements required by multiple users' applications can be accommodated from one data source without changing the physical organizations of the data in the database. Furthermore insertions, deletions and modifications of data elements can be done without affecting all application programmes that use the same database. This greatly reduces flexibility and programme maintenance cost considerably. With a DBMS, the user need not be concerned with the physical location organization and the procedure for accessing information needed for his particular purpose.

1.2 <u>Data Models</u>

In the context of data base systems, a data model is a term used to denote any formally definable class of data structures, which can be used as the basis for the design and development of various data processing applications. According to McGee, a data model is a way of viewing data, it provides a basis for the construction of a DBMS. Three kinds of important data models have been proposed. They are:

(1) The Hierarchical data model (Bleir[10]).

(2) The Network data model (Bachman[3]; CODASYL[12], 1971).

(3) The Relational data model (Codd[13]).

Hierarchical data models are embodied in the form of IBM's Information Management System (IMS) and MRI's System 2000(S2K). In the Hierarchical approach the data base is represented by an ordered tree, the nodes of which usually correspond to entity types, which are represented by tables of data, and arcs between the nodes correspond to the functional relationships between the tables.

The importance of Network approach grows after it is put forth by the Data Base Task Group (DBTG) of the Conference on Data System Languages (CODASYL) in 1976. In Network approach, data is represented by records and links, and the relationships between records are called as sets. The data structure is represented by a network structure.

1.3 The Relational Model

The concept of relational model of data was first proposed by Codd [13] in 1969. In this system one views the database as a set of relations, where the term relation has been derived from the mathematical definition of relations. Conceptually, a relation can be viewed as a table in which each row corresponds to records of files known as entity (or tuple) and each column corresponds to field of records known as an attribute. There exist a set of possible values associated with each attribute in a relation, called the domain of that attribute.

Formally, a relation can be defined as follows (Date ,1981,[15],p-84):

Given a collection of a sets $D_1, D_2, \ldots \ldots D_n$, (not necessarily distinct), a relation R defined over the set $\{D_1, D_2, \ldots\ldots, D_n\}$ is a subset of the cartesian product $D_1 \times D_2 \times \ldots\ldots \times D_n$. That is, R is a set of ordered n-tuples each of the form $(d_1, d_2, \ldots\ldots, d_n)$ where $d_i \in D_i$. Each element of R is called a tuple of R. An attribute is a name assigned to a domain of a relation. While the domains of a relation need not be distinct, the attribute names assigned to them must be unique with in the relation.

A n-ary relation denoted by $R(A_1,A_2,\ldots\ldots\ldots\ldots,A_n)$ is also defined (Delobel [17]) as:

- a set of attributes $\{A_1,A_2,\ldots\ldots,A_n\}$,
- a sequence of domains $Dom(A_i)$ for $i=1,2,\ldots,n$ which defines the potential value for each attribute.
- a predicate denoted by $|R(A_1,A_2,\ldots\ldots\ldots,A_n)|$ which represents the potential evaluation of the predicate when values are assigned to the attributes according to their domains.

Suppose $A_1,A_2,\ldots\ldots\ldots,A_n$ are the names of the domains $D_1,D_2,\ldots\ldots\ldots,$ and D_n respectively of a relation R, then we use notation (1.3.1) for R.

$$R(A_1,A_2,\ldots\ldots\ldots,A_n) \qquad 1.3.1$$

The attribute set of R is defined as:

$$U(A_1,A_2\ldots\ldots\ldots\ldots,A_n) \qquad 1.3.2$$

We will use (1.3.3) to designate the relation R on the set of attributes U.

$$R(U) \qquad 1.3.3$$

The structure of relation is sometimes called as the <u>intension</u> (Scheme) and the contents of a relation is referred to as the extension. The contents of a relation may vary from time to time. That is, tuples may be modified, deleted from a relation, or/and inserted in a relation. The contents of a relation in a particular time is called as its <u>instance</u>. Figure 1.1a indicates a relational schema for a medical database consisting of four relational schemes HOSPITAL (CODE, NAME, ADDRESS, # OF BEDS), DOCTOR (DOCTOR #, NAME, SPECIALIZATION), WARD (WARD CODE, NAME, # OF BEDS) and STAFF (EMPLOYEE #, NAME, DUTY, SALARY). An instance of the schema is shown in figure 1.1b.

Let R(U) be a relation on the set of attributes U and let W be a subset of U, then W is a candidate key of R if it can uniquely identify the tuples of R and no proper subset of W has this property. A relation may have more then one candidate key. An attribute is said to be <u>prime</u> if it appears in any candidate key of the relation, otherwise it is called a <u>non-prime</u> attribute.

Conventions: Upper-case letters A,B,C,...... from the start of alphabet represent single attribute; upper-case letters ......X,Y,Z from the end of alphabet represent sets of attributes; and lower-case letters r,s,t,..... from the end of alphabet represent tuples of a relation.

A Relational Schema and its instance

a) An example of a relational schema (consisting of four relational schemes)

HOSPITAL (CODE,NAME,ADDRESS,# OF BEDS)={(H1,AM,DL,500), (H2,RL,ND,250), (H3,JH,SD,50)}

DOCTOR (DOCTOR #,NAME,SPECIALIZATION) = { (D1,RT,CN), (D2,SK,SY), (D3,NM,OP), (D4,AN,AT), (D5,AP,FP) }

WARD (WARD CODE, NAME, # OF BEDS) = { (W1,XC,45), (W2,YD,30), (W3,ZA,56), (W4,UE,25), (W5,VF,85) }

STAFF (EMPLOYEE #,NAME,DUTY,SALARY) = { (10,DG,XX,650), (25,FH,XY,756), (15,PA,NW,350), (23,RG,SU,215), (26,DM,WK,234), (45,SB,SN,542) }

Figure 1.1a A medical database schema

b) An instance of the schema of the part "a".

HOSPITAL (CODE, NAME, ADDRESS, # OF BEDS)

H1	AM	DL	500
H2	RL	ND	250
H3	JH	SD	50

DOCTOR (DOCTOR #, NAME, SPECIALIZATION)

D1	RT	CN
D2	SK	SY
D3	NM	OP
D4	AN	AT
D5	AP	FP

WARD (WARD CODE, NAME, # OF BEDS)

W1	XC	45
W2	YD	30
W3	ZA	56
W4	UE	25

STAFF (EMPLOYEE #, NAME, DUTY, SALARY)

15	PA	NW	350
23	RG	SU	215
26	DM	WK	234
45	SB	SN	542

Figure 1.1b

1.4 Operations on Relation

One of the main advantages of the relational approach is that, if the relations are created to confirm certain mathematical constraints, then the relations can be manipulated mathematically. The manipulation is accomplished through the data manipulation languages (DML). There are a number of ways in which relations can be manipulated. Several relational systems provide a DML which is based on relational algebra, other systems provide languages based on relational calculus. The relational algebra suggested by Codd [13], is a collection of set operations out of which the two most important operations i.e. Projection and Join are discussed below.

Let R be a relation defined on the set of attributes $U = \{A_1, A_2, \ldots\ldots, A_n\}$. For any $W = \{A_1, A_2, \ldots \ldots, A_m\}$, or $m \leq n$ that is $W \subseteq U$, the projection of R on W is denoted by R[W] and is defined as:

$$R[w] = \{(a_1, a_2, \ldots\ldots\ldots, a_m) \mid (a_1, a_2, \ldots\ldots, a_n) \in R\}$$

In otherwords, we can think of the projection of R on W as the operation that takes the relation (instance) represented by R, then delete all

The Projection operation

If An instance of a relation R (A, B, C, D, E) is

R (A,	B,	C,	D,	E,)
a_1	b_3	c_2	d_1	e_3
a_1	b_2	c_3	d_1	e_4
a_2	b_4	c_1	d_3	e_4
a_3	b_1	c_3	d_2	e_1
a_3	b_2	c_4	d_4	e_2

then the projection of R on (A, B, D) is R[A, B, D], given by:

R [A,	B,	D]
a_1	b_3	d_1
a_1	b_2	d_1
a_2	b_4	d_3
a_3	b_1	d_2
a_3	b_2	d_4

Figure-1.2

columns except those labelled by attributes in W and finally identify the common tuples, as shown in figure1.2.

The Join operation which in some sense is inverse to the projection operation, in fact connects attributes of different relations together. The Join (natural Join) of a relation R(X, Y) with a relation S(Y, Z) is denoted by R*S and is defined as:

$$R*S = \{(x,y,z) \mid (x,y) \in R \text{ and } (y,z) \in S\} \qquad 1.4.2$$

Figure 1.3 indicates an example of join operation.

1.5 Data Dependencies

In relational database model, conceived by Codd, one views the database as a collection of relations, where each relation is a set of tuples over some domains of values. One notable feature of this model is its being almost devoid of semantics. A tuple in a relation represents a relationship between certain values, but from mere syntactic definition one knows nothing about the nature of this relationship, not even whether it is one-to-one or one-to-many relationship. One approach to remedy this deficiency is to devise means to specify the missing semantics. This semantic specification is often called semantic or integrity

The Join operation

If the instances of two relations R(A,B,C) and S(C,D) are

R(A,	B,	C)
a_1	b_3	c_2
a_1	b_2	c_3
a_2	b_4	c_1
a_3	b_1	c_4

S(C,	D)
c_1	d_2
c_2	d_2
c_3	d_3
c_4	d_1

then the natural join of R and S (i.e. R*S) is given by:

R*S (A,	B,	C,	D)
a_1	b_3	c_2	d_2
a_1	b_2	c_3	d_3
a_2	b_4	c_1	d_2
a_3	b_1	c_4	d_1

Figure-1.3

constraints, since they specify which databases are meaningful for the application and which are meaningless. The constraints are called data dependencies or simply dependencies in database systems.

The study of dependencies began in 1972 with the introduction by Codd[14] of the Functional Dependencies. After the introduction, independently by Fagin [19] and Zaniole [34] in 1976, of Multivalued Dependencies, the field becomes chaotic for a few years in which various researchers introduced many new classes of dependencies.The situation has situated stabilized since 1980 with the introduction of Embedded Implicational Dependencies (EIDs). Essentially, EIDs are sentences in first order logic, stating that if some tuples fulfilling certain equalities, then either some other tuple must also exist in the data base or some values in the given tuples must be equal. The cnass of EIDs seems to contain most of the previously studied classes of dependencies. Recently De Bra and Paredaens[16] considered afunctional dependencies, which are not functional dependencies. In the fonlowing subsection we give the basic definitions of various kinds of dependencies and in the next chapter we will discuss the properties of some of the important data dependencies.

1.5.1 Functional Dependencies

Functional dependencies (abbr. as FDs) form a family of constraints, the properties of which have been studied extensively by Armstrong[2], Beeri et al[6], Fagin[21] and Nicolas[27]. We give the definations of FDs below.

A functional dependency is a sentence denoted by f:X->Y, where X and Y are sets of attributes. An FD f:X->Y holds in a relation R(U) where X and Y are subsets of U, if for every tuple u and v of R, u[X]=v[X] implies u[Y]=v[Y], i.e. the relation R obeys the FD f:X->Y if for every two tuples of R, which have the same projection on X also have the same projection on Y. Given f:X->Y, we say that X functionally determines Y, or Y is functionally dependent on X. We usually write the FD f:X -> Y simply as X -> Y.

FDs can also be represented by first-order logic as shown by Nicolas [27]. Consider the relation R(U), where U={A,B,C,D}. The FD is represented by the sentence:

$$(\forall\, abc_1c_2d_1d_2)((Pabc_1d_1 \wedge Pabc_2d_2) \Rightarrow (c_1=c_2))$$

Where $(\forall\, abc_1c_2d_1d_2)$ is a shorthand for $\forall a \forall b \forall c_1 \forall c_2 \forall d_1 \forall d_2$ i.e. each variable is universally quantified and P is a relational symbol.

1.5.2 Multivalued Dependencies (abbr. MVDs)

The concept of functional dependencies is not sufficient to capture the various types of relationships that exist in relations. It is possible that the values of the attributes in a set Y depends only on the values of the attributes in the set X, but there exist more than one Y-value for a given X-value. Such a relationship is not a functional dependency. Hence the concept of multivalued dependency (MVD) was introduced independently by Fagin [19] and Zaniolo [34] to describe such relationships.

Definition:- Let R(U) be a relation schema and let Y be a subset of U, for each subset X of U and for each X-value x, we define

$Y_x = \{ y \mid$ for some tuple $t \in R$, $t[X]=x$ and $t[Y]=y \}$, where t[X] is the X-component of the tuple t in R.

i.e. Y_x is a function that gives for each X-value, the set of Y-values that appear with this X-value in tuples of the relation.

A multivalued dependency g, on a set of attributes U is a statement g: X->->Y, where X and Y are subsets of U. Let Z be the complement of the union of X and Y in U.

A relation schema R(U) obeys the MVD g: X->->Y, if for every XZ-value xz, that appears in R, we have $Y_{xz} = Y_x$. In other words, the MVD g is valid in R if the set of Y-values that appears in R with a given x, also appears with every combination of x and z in R.

Fagin [19] defined the MVD as: let R be a relation schema over U and X and Y are subsets of U. Let Z be the complement of the union of X and Y in U i.e. Z=(U-XY). Then the MVD X ->-> Y holds in R if for all tuples r_1 and r_2 in R $r_1[X] = r_2[X]$, then there exist tuples r_3 and r_4 such that

(i) $r_3[X] = r_1[X]$, $r_3[Y] = r_1[Y]$ and $r_3[Z] = r_2[Z]$;

(ii) $r_4[X] = r_2[X]$, $r_4[Y] = r_2[Y]$ and $r_4[Z] = r_1[Z]$.

Like FDs, the MVDs can also be expressed in first-order logic. For example assume that U={A,B,C,D,E}. Then the MVD AB->->CD holds for a relation over U if the sentence

$$(\forall\, abc_1c_2d_1d_2e_1e_2)\ ((Pabc_1d_1e_1) \wedge (Pabc_2d_2e_2) \Rightarrow Pabc_2d_3e_1)$$

holds, where P plays the role of the relation symbol.

1.5.3 Hierarchical Dependencies

A hierarchical dependency (abbr. HD) is a sentence denoted by $X{:}Y_1|Y_2|\ldots\ldots\ldots\ldots\ldots|Y_K$ where $X, Y_1, Y_2, \ldots\ldots\ldots\ldots, Y_k$ are disjoint sets of attributes.

A relation $R(X, Y_1, Y_2, \ldots\ldots\ldots\ldots, Y_k)$, where $X, Y_1, Y_2, \ldots\ldots\ldots\ldots$ and Y_k are disjoint sets of attributes, obeys the HD if for every X-value x we have

$$R(X, Y_1, Y_2, \ldots\ldots, Y_K) = R[X, Y_1] * R[X, Y_2] * \ldots\ldots\ldots \ldots\ldots * R[X, Y_k]$$

which expresses the decomposability of the relation of R over $\{X, Y_1, \ldots\ldots\ldots\ldots, Y_k\}$ into k projections.

We shall say that a HD is a total hierarchical dependency (also called as a generalized multivalued dependency (GMVD)) if X is the empty set, since in this case the projection of R over the set attributes $\{Y_1, Y_2, \ldots\ldots\ldots Y_k\}$ is the projection of R respectively on $Y_1, Y_2, \ldots\ldots$, and Y_k.

1.5.4 Join Dependencies

The MVDs characterize the lossless decomposition of a relation into two projections. However

MVDs are inadequate for expressing the conditions under a lossless decomposition of a relation into more than two projections. Join dependencies were introduced to characterize this kind of lossless decomposition.

Definition:- Let R(U) be a relation over an attribute set U, and $X_1, X_2, \ldots\ldots\ldots, X_n$ are subsets of U such that $\bigcup X_i = U$, a n-join dependency (abbr. n-JD) is a sentence denoted as $* [X_1][X_2]\ldots\ldots\ldots.[X_n]$, also denoted as $*$ [U]. The relation R is said to satisfy this n-JD if

$$R = \mathop{*}_{i=1}^{n} R[X_i]$$

i.e if R is the join of its projections $R[X_1], \ldots\ldots.$ $\ldots.., R[X_n]$. It follows that this JD $*$ [U] holds for the relation if and only if R contains each tuple t for which there are tuples $t_1, \ldots., t_n$ of R such that $t_i[X_i] = t[X_i]$ for each i $(1 \leqslant i \leqslant n)$.

An n-JD characterizes exactly the lossless decomposition of R into n projections. The JD can express multivalued dependencies and total hierarchical dependencies in a unified way. This follows directly from their definitions. A multivalued dependency $X \to\to Y$ can be represented by a 2-JD $*[XY][YZ]$ and a total hierarchical dependency $X:Y_1 | Y_2 | \ldots\ldots\ldots. | Y_k$ can be represented by the k-JD

$$* [XY_1][XY_2]\ldots\ldots\ldots[XY_K].$$

1.6 Normalization and Normal Forms

The notion of normalization in relational database was first presented by Codd[14]. He observed that certain relations have structural properties those are undesirable for describing data bases. These undesirability stem from the fact that some attributes are related to each other in certain ways. For example consider the relation SUPP(SUPPLIER, TOWN,POPULATION). Its intended meaning is that, whenever a tuple say (s,t,p) occurs in this relation, it means that " supplier s is located in the town t whose population is p." The relation scheme in fact leads to the following data manipulation anomalies. First, notice that the population of a given town must appear as many times as there are suppliers located in that town (data redundancy). Thus if the population of a town has to be updated, all the tuples in which it occurs have to be retrieved in order to update consistently the population of the town (updating anomaly). Now, if the last supplier located in a given town is deleted, then the population of this town is lost (deletion anomaly). Conversely, the population of a town can be recorded only when one knows at least one supplier located in that town (Insertion anomaly).

To avoid data manipulation anomalies attempts have been made to introduce schemes with no undesirable structural properties for describing data-base. This consideration led to Codd [14] to define a process known as normalization, which consisting of converting a relational schema into another form that stores the same data but in different format and ensure the removal of undesirable anomalies and redundant attributes from the relational schemes. In [14] Codd has discussed the normalization of relations, which is based on a series of four normal forms which are, First Normal Form, Second Normal Form, Third Normal Forms and Boyce-Codd Normal Form.

Later in 1977, Fagin[19] discovered that even by putting a schema in Boyce-Codd Normal Form, not all the anomaly problems necessarily disappear. This led him to propose a new normal form called Fourth Normal Form. Forth Normal Form is defined in terms of functional and multivalued dependencies alone. It has shown by Fagin [19] that the concept of multivalued dependency is intimately related to the join dependencies. For example, if U and V are the subsets of attributes of a relation R and if W is the set of attributes of R not in U or V, then the MVD U->->V holds in R if and only if R is the join of its projections R[UV] and R[UW] i.e. if the JD *[UV,UW] holds in R. Hence multivalued dependencies are correspond to "2-way decompositions of a relation. But Aho, Beeri and Ullman [1] have given a surprising example to show that a relation can be the join of three of its projections, without this join being the result of cascading 2-way projections. Fagin has introduced another normal form known as Project and Join Normal Form (PJ/NF) and have shown that, because of the above property the PJ/NF is stronger then the 4NF. These normal forms are discussed in the following subsection.

1.6.1 Normal Forms

The concept of functional dependencies and multivalued dependencies play significant roles in the theory which governs the decomposition of relations into subrelations in normal forms.

To show how certain undesirable dependencies create problems, we will discuss the concept of partial functional dependencies, full functional dependencies, key dependencies and transitive dependencies mentioned by Codd [14]. We will also discuss Fagin's notion of nontrivial multivalued dependencies.

Let R be a relation schema defined over the set of attributes U. We say that Y is fully dependent on X in R if

(i) X and Y are two disjoint subsets of attributes of relation R.

(ii) X->Y, and

(iii) Y is not functionally dependent on any proper subset of X.

If the condition (3) is not satisfied then we say Y is partially dependent on X in relation R.

If K is a subset of U, then we say that K is a key (of the relation schema) if the FD K -> U is in the schema R, and if there is no proper subset L of K such that the FD L->U is also in the schema. We call such a functional dependency K->U a key dependency of R. That is the dependency K->U is a key dependency if it is a full FD in R.

<u>First Normal Form</u> (1NF):

A relation schema R is said to be in

1NF if and only if all the underlying domains of each attribute of R contain atomic values.

Second Normal Form (2NF):

A relation schema R is said to be in 2NF if

(i) it is in 1NF, and

(ii) every non-prime attribute of R is fully dependent on each candidate key of R.

Third Normal Form (3NF):

To define the third normal form we need to define "Transitive Dependency".

Given a relation schema R, suppose that X,Y, and Z are three distinct collection of attributes of R, and if the following conditions are true:

(i) $X \to Y$

(ii) $Y \not\to X$

(iii) $Y \to Z$

then it follows that $X \to Z$ and $Z \not\to X$.

Here Z is said to be transitively dependent on X in the relation R.

A relation schema R is said to be in 3NF if,

(i) it is in 2NF, and

(ii) every non-prime attribute is non-transitively dependent on each candidate of R.

Boyce-Codd Normal Form (BCNF):

The BCNF can be defined in the following three distinct (but equivalent) ways:

(1) A 1NF relation schema R with attributes U is said to be in BCNF if, for each non-trivial FD $X \rightarrow Y$ in R, the FD $X \rightarrow U$ is also in R.

(2) A 1NF relational schema R with attributes is said to be in BCNF if $G \vdash f$ i.e. if f can be derived from the set G, for each FD f in R, where G is the set of key dependencies in R.

(3) A 1NF relation schema R with attributes is said to be in BCNF if, for each FD f in R, there is a key dependency $K \rightarrow U$ in R such that $K \rightarrow U \vdash f$.

Thus the BCNF states that every set of attributes which has another attribute functionally dependent upon it in a relation schema R, must be a candidate key of R.

Fourth Normal Form (4NF):

The concept of "trivial multivalued

dependencies" proposed by Fagin[19], is needed in describing the forth normal form relations.

Given a relation R(U) where U = {X,Y}, then the multivalued dependencies X->->Y and X->-> θ, where θ is the null set, are necessarily hold for R. These are called trivial multivalued dependencies.

We now define the 4NF in the following ways:

(1) A 1NF relation R with attributes U is said in 4NF if, for each non-trivial MVD X->->Y holds for R, then so does the functional dependency X -> U holds in R.

(2) A 1NF relation schema R with attributes U is said to be in 4NF if, G $\vdash$ m for each MVD m in R, where G is the set of key dependencies in R.

(3) A relation schema R with attributes U is in 4NF if, for every MVD m in R there is a key dependency K->U of R such that (K->U) $\vdash$m.

Thus a relation scheme R is said to be in Fourth Normal Form if, every MVD in R is a result of keys of R.

<u>Project-Join</u> <u>Normal</u> <u>Form</u>: (PJ/NF)

We define the PJ/NF in the following ways:

1) A 1NF relation schema R with attributes U is in PJ/NF if $K \vdash j$ for each JD j in R, where K is the set of key dependencies of R.

2) A 1NF relation schema R with attributes U is in PJ/NF if, for each JD j in R, there is a key dependency $K_i \rightarrow U$ in R such that $(K_i \rightarrow U) \vdash j$.

II. IMPLICATION PROBLEMS FOR DEPENDENCIES

1 Introduction

The theoretical bases for data dependencies in a relational data model are discussed in this chapter in details. A set of axioms i.e. inference rules for the family of functional dependencies has been explained and it has been shown that these axioms are complete for this family. Also a complete set of inference rules for multivalued dependencies has been presented in this chapter. It has been stated that the combination of inference rules for Fds and MVDs is not sufficient for the family of FDs and MVDs, and thus additional rules (FD-MVD rules) have been given to complete the set of rules for FDs and MVDs. Also we have presented a complete set of inference rules for the set of join dependencies in this chapter. Furthermore the closure of a set of dependencies and also for a set of attributes and various types of covers of a set of FDs are also discussed in this chapter.

.2 The Inference Rules for Data Dependencies

The most important problem for dependency theory is the implication problem i.e. the problem of deciding for a given set of dependencies G and a dependency g, whether $G \models g$ i.e. whether G logically implies g. A dependency g is said to be logically implied by a set of dependencies G, if g is valid in every relation which obeys all the dependencies in G. In other words g is logically implied by G, if there does not exist any counter example relation which obeys all the dependencies in G but does not obey g. The reason for prominence of the problem is that an algorithm for testing implication of the dependencies enable us to test whether two given sets of dependencies are equivalent, or a given set of dependencies is redundant. Even though the significant of implication problem was not yet clear in 1974, it was studied by Armstrong [2] apparently out of mathematical interest. Armstrong characterized implication of functional dependencies by using an axiom system where an axiom system consists of axiom schemes and a set of inference rules. A derivation of a dependency g from a set of dependencies G, denoted by $G \vdash g$, is a sequence $g_1, g_2, \ldots\ldots\ldots, g_n$ where g_n is either an instance of the axiom scheme or follows from the preceding dependencies in the sequence by one of the inference rules.

A set of inference rules is said to be

complete for a family of dependencies if for each set G of dependencies from the family, the dependencies that are implied by the set of dependencies G are exactly those, that can be derived from it using the set of inference rules.That is a set of inference rules is said to be complete if G|-g entails G|=g. The concept of completeness of a set of inference rules is of prime importance in a system where inference rules are these are used. If a complete set of rules is used then only the database designer can be assured that he has a complete knowledge of all dependencies that hold in a given database. A complete set of inference rules is said to be minimal if no proper subset of it is com plete. Armstrong's rules for functional dependencies are complete is one of the basic assumption in the works on functional dependencies. For multivalued dependencies a complete set of inference rules is given by Fagin[19] and Zaniolo[34] in somewhat restricted manner. Beeri et al [6] have removed these restrictions and presented a general complete set of inference rules for FDs and MVDs. Mendelzone [25] further investigated about the independence and redundancy of these rules and has given a minimal complete set of inference rules for multivalued dependencies. After the introduction of join dependencies by Rissenen[29], a complete axiomatization of full join dependencies is presented by Sciore [31]. Detailed discussion for the inference rules for FDs and MVDs are given in the following subsections.

2.2.1 Inference Rules for Functional Dependencies

Axiomatization of functional dependencies was studied by Armstrong [2]. He has presented a set of axioms governing the set of functional dependencies. It has been proved [2, 6] that this set of axioms is complete for the family of functional dependencies. The completeness of Armstrong's axioms for FDs is an important basis for research in this area (including the present dissertation work).The complete set of axioms for the family of functionally dependencies is presented below.

FD Rules:

In the following rules, X,Y,Z and W are arbitrary subsets of U, where U is the set of all attributes. We write XY for the union of the two arbitrary sets X and Y.

FD1 (Reflexivity): If $Y \subseteq X$ then X->Y.

FD2 (Augmentation): If $Z \subseteq W$ and X->Y then XW->YZ.

FD3 (transitivity): If X->Y and Y->Z then X->Z.

Other used rules :-

FD4 (Pseudotransitivity): If X->Y and YW->Z then XW->Z.

FD5 (Union): If X->Y and X->Z then X->YZ.

FD6 (Decomposition): If X->YZ then X->Y and X->Z.

FD7 (Projectibility): If X->Y holds in R(U) and $X \subseteq W \subseteq U$ then X->Y holds in R[W].

FD8 (Reverse projectibility): If X->Y holds in a projection of R(U) then X->Y holds in R(U).

If A and B are attributes of a relation R, then by applying the axiom FD1 to X = {A, B} we get AB->AB, AB->A, AB->B, A->A and B->B.

Axiom FD2 means that, knowing f:X->Y, we can construct another functional dependency, say g: XA -> Y, where the attributes appearing on the left side of g consisting of the attributes of X plus some other extraneous attribute A, whose values have no effect on the values of Y selected by g.

For axiom FD3, if the FDs f:X->Y and Y->Z holds in a relation R then the dependency h:X->Z also holds in R.

In the above set of axioms, the Axioms FD1-FD3 are sufficient and the other additional axioms i.e. FD4-FD8 are implied by the first three axioms. As an example, Axiom FD4 can be derived from the axioms FD1-FD3 as follows.

As our assumption we have f: X -> Y and YW->Z. Now from f and Axiom FD2 we get h: XW -> YW. By applying axiom FD3 to h we can derive an FD XW->Z, completing the claim. Similarly it is easy to show that the other axioms can also derive from the first three axioms.

2.2.2 Inference Rules for Multivalued Dependencies

A set of rules for multivalued dependencies has been presented by Beeri et al [6] and it has been proved that the given set is complete for the family of multivalued dependencies. The complete set of inference rules is explained below. In the rules, X,Y,Z and W are arbitrary sets of attributes. We use XY for the union of two sets X and Y.

MVD Rules :-

MVD0 (complementation): If U=XYZ and $Y \cap Z \subseteq X$, then X->->Y iff X->->Z.

MVD1 (Reflexivity): If $Y \subseteq X$ then X->->Y.

MVD2 (Augmentation): If $Z \subseteq W$ and Y->->X then YW->->XZ.

MVD3 (Transitivity): If X->->Y and Y->->Z then X->->(Z-Y).

<u>Other useful rules are</u>:-

MVD4 (Pseudotransitivity): If X->->Y and YW->->Z then XW->->(Z-YW).

MVD5 (UNION): If X->->Y AND X->->Z then X->->YZ.

MVD6 (Decomposition): If X->->Y and X->->Z then X->->Y Z, X->->(Y-Z) and X->->(Z-Y).

The validity of these rules have been proved by Fagin [19] and Beeri et al in [5]. Beeri et al have proved that the inference rules MVD0-MVD3 are complete for multivalued dependencies. Mendelzone [25] has investigated about the independence and redundancy of these rules and proved that the set {MVD0,MVD1,MVD3} forms a minimal complete set of inference rules for Multivalued Dependencies.

2.2.3 <u>Mixed inference rules for FDs and MVDs</u>

In the previous two subsections we deal with the implication problems of FDs and MVDs only i.e given a set F of FDs whether any other FD f is implied by F, and given a set G of MVDs whether any other MVD g is implied by G. The problem of implication of additional dependencies, that are implied by the combination of FDs and MVDs i.e. by FUG, has been discussed thoroughly by Beeri et al [6], and the following rules have been proposed.

<u>Mixed</u> <u>Rules</u>:-

FD-MVD1: If X->Y then X->->Y.

FD-MVD2: If X->->Y and Z->Y', where $Y \supset Y'$ and Y and Z are disjoint then X->Y'.

Zaniolo [35] has pointed out that the rule FD-MVD2 has been defined in a restricted manner and presented an alternation and simple rule called <u>mixed transitivity</u> rule for FDs and MVDs, which is defined as:

FD-MVD3 (Mixed transitivity): If X->->Y and Y -> Z then X -> (Z - Y).

Zaniolo has shown that the set {FD1,FD2,FD3,MVD0, MVD1,MVD2,MVD3,FD-MVD1,FD-MVD3} is a complete set of inference rules for the combination of the FDs and MVDs.

2.2.4 Inference Rules for Join Dependencies

A complete set of inference rules has been proposed by Sciore[31] and has been discussed below. In the following rules R and S represents two relational schemas defined over the attribute sets U and V respectively. We use the notation J|-D for the derivation of a dependency D from a set of dependency J, and Ø for the null set.

JD-Rules

JD0: Ø|-*[X] for any set $X \subseteq U$. Which states that the dependency *[X] is a trivial dependency in R.

JD1: (Covering rule).

[S]|-[R] if U=V and R covers S i.e. if for every subset Y of V there exist a subset X of U such that $Y \subseteq X$.

To simplify the use of covering rule, a set of four special cases have been given and are:

JD1a: *[S]|-*[S,Y] if $Y \subseteq V$. (add a set)

JD1b: *[S,Y,Z]|-*[S,YZ]. (replace a set by their union)

JD1c: $*[S_1,S]$|- $*[S_1A,S_2]$ (add an attribute to a set)

JD1d: $*[S_1,S_2,\ldots,S_k] \vdash *[S_1,\ldots,S_{k-1}]$

if $S_k \subseteq S_j$ for some $j \neq k$.

JD2: (substitution rule).

$\{*[S,X].*[R]\} \vdash *[S,R]$ if $U=X$.

JD3: (projection rule).

$*[S,YA] \vdash *[S,Y]$ if $A \in V$.

The rule JDO is an axiom that allows us to infer only trivial dependencies. The rules JD1 and JD3 allow us to infer from one given dependency another dependency that is less informative then the one that is given, where as the rule JD2 allows us to combine two dependencies to yeild a third dependency that is more informative then either one of the given dependencies. It has been proved by Sciore [31] that the set { JDO, JD1, JD2, JD3} forms a complete set of inference rules for the set of JDs in a relational schema.

2.3 The Closures of Dependencies

The definition of the closure of a set of dependencies and the closure of a set of attributes with respective to a given set of dependencies and also the definitions of various types of coverings of a set of functional dependencies are presented below.

Definition 2.3.1: If G is the union of the set F of

FDs and the set M of MVDs i.e. $G=F\cup M$, then the closure of G denoted by G^+ is the set of FDs and MVDs that can be derived from the repeated application of FD rules, MVD rules and their mixed rules. Similarly if F is the set of FDs over a set of attributes U, then the closure of F, denoted by F^+, is defined to be the set of all FDs that can be obtained by the successive application of the rules FD1, FD2 and FD3 on the set F.

<u>Definition 2.3.2</u>: The closure of a set of attributes X with respect to a set of dependencies G defined over a set of attributes U, is denoted by X^+ and is defined as the set of all attributes that are found to be functionally dependent on X, which are implied by G.

i.e. $$X^+ = \{A \mid X \rightarrow A \in G^+\}.$$

<u>Definition 2.3.3</u>: A set of FDs G is said to be a cover of another set of FDs F or G is said to be equivalent to F, if and only if $F^+ = G^+$.

<u>Definition 2.3.4</u>: A set of FDs G is said to be a non-redundant cover of another set of FDs F, iff $G^+=F^+$ and there does not exist a set of FDs H such that $H \subset G$ and $H^+=G^+$.

<u>Definition 2.3.5</u>: A set of FDs G is said to be a minimum cover of a set of FDs F, iff $F^+=G^+$ and there

does not exist a set H with fewer FDs then G such that $H^+=G^+$.

Definition 2.3.6: An FD f is said to be redundant in a set of FDs F, iff $F^+=(F-f)^+$.

Definition 2.3.7: Let F be a given set of FDs and $f:X \rightarrow A$ be an FD in F^+. An attribute B is said to be extraneous or redundant in X with respect to F if, $B \in X$ and $[(X-B) \rightarrow A]$ is in F^+.

2.4 Chapter Summary and Remarks

The implication problem for data dependencies in a relational database have been explained throughly in this chapter. A complete set of inference rules for FDs has been presented and also the additional rules for FDs which are required for the manipulation of other FDs are given. A complete set of inference rules for MVDs has been explained and it has been pointed out that the combination of rules for FDs and MVDs is not complete for the family of FDs and MVDs and hence additional rules known as Mixed Rules have been given to complete the set of rules for FDs and MVDs. Also a complete set of inference rules for the set of JDs are discussed. Furthermore the various types of covers of a set of FDs are explained in this chapter.

III. A MEMBERSHIP ALGORITHM FOR FUNCTIONAL DEPENDENCIES

3.1 Introduction

The basic concept underlying the search for suitable normal forms may be described as an attempt to develop a design methodology for relational database schema. The synthesis and decomposition approaches are the two alternative ways for obtaining a normalized database schema. There are several synthesis algorithms for designing a database schema when only functional dependencies are given [0,9,20], and all of them use some sorts of cover of the given set of functional dependencies. A recent paper [4] describes a synthesis algorithm for providing a normalized database schema when both functional dependencies and multivalued dependencies are given. All these algorithms are built around a membership test for functional and multivalued dependencies, where the membership problem is to determine whether a given set of dependencies G implies another dependency g. In decomposition approach, a part of the problem is to decide whether a nontrivial functional or multivalued dependency holds in a relation scheme. This decision problem can be solved by applying a membership algorithm for dependencies as shown in [23]. Thus an efficient membership algorithm is an important tool for designing normalized database schemas.

Both functional and multivalued dependencies have inference rules as described in chapter-II, that can be used to infer a dependency g from a given set of dependencies G if and only if g is implied by G. In [4] Bernstein have used these rules to develop a linear time algorithm for functional dependencies. A similar approach used by Beeri [4] to devise an $o(\|G\|^4)$-time membership algorithm for functional and multivalued dependencies, where $\|G\|$ is the size of description of G. A refinement of this algorithm based on an appropriate data structure has an $o(\min\{k^2|U|, \|G\|^2\})$ running time, where U is the set of all attributes and k and $|U|$ are number of dependencies in G and number of attributes in U respectively is given in [23].

In this chapter we give a membership algorithm for functional dependencies and show that this algorithm is faster then the previous algorithms and also it requires a simple data structure for implementation.

We have organized this chapter as follows: In section two we describe the method for developing a membership algorithm for FDs and give a linear-time algorithm. In section three this algorithm has been modified by using a simple data structure for efficient implementation of the algorithm . In section four we have analyzed the implementation of the algorithm. In section five the application of the membership algorithm and section six carries some concluding remarks.

3.2 The Membership Algorithm

The membership problem for functional dependencies says that "Given a set of FDs G and an FD g, determine whether $g \in G^+$ i.e. whether g is in closure of the given set of FDs ". We start with designing a simple algorithm for the membership problem for FDs and refine it by using a simple data structure for implementation purpose.

The membership algorithm can be solved by computing the closure of the given set of FDs G i.e. G^+ by using the complete set of inference rules for FDs. But the computation of G^+ is a time consuming job, because even if G is very small the set of dependencies in G^+ will become very large. It has shown that the dependency $X \to Y$ is in G^+ if $Y \in X^+$, where X^+ is the closure of the set of attributes X with respect to G and since the computation of X^+ requires time proportional to the length of all dependencies in G as shown by Beeri et al [5], we will follow this method and develop an algorithm for which the implementation time can be reduced considerably . The algorithm is given in figure 3.1 and is described below.

In this section we consider the given FD $g: X \to Y$ and a set of FDs G. We assume that X and Y are disjoint, since $X \to Y$ is a consequence of G if and only if $X \to (Y-X)$ is a consequence of G.

The algorithm given in figure-3.1 i.e. the Algorithm-1 uses the procedure FIND(Y) that computes Y' which is a subset of Y, and is obtained by eliminating the attributes from Y which are found to be functionally depending on X with respect to the set of FDs G. If Y' is found to be a null set then, all the attributes in Y are depending on X with respect to G which implies that $X \to Y$ is in the closure of G i.e. $X \to Y \in G^+$. To compute Y' we will follow the procedure given below.

Now $(Y-Y')$ is depending on X, hence is a subset of the closure of set of attributes X i.e.

ALGORITHM-1

Input: A set G of m FD's on attributes $\{A_1, A_2, \ldots\ldots\ldots, A_n\}$ and an FD $g: X \rightarrow Y$.

Output: 'YES' if $g \in G^+$; 'NO' if $g \notin G^+$.

<u>Data structures:</u>

1. Attributes are represented by integers between 1 and n.
2. FD's are represented by integers between 1 and m.
3. DEPEND is a set of attributes found to be functionally depending on the set X so for.
4. Y' is a subset of attributes of the set Y, which are not yet found to be functionally depending on X so for.
5. QUEUE is a set of FD's whose left hand sides are found to be asubsets of DEPEND so for.

ALGORITHM:-

```
procedure FIND(Y):
 begin
        make QUEUE empty;
             DEPEND = X;
                 Y' = Y;
```

```
(4)        put every dependency of G with a left
           hand side a subset of DEPEND, on QUEUE;
(5)       while ((QUEUE is not empty) AND (Y = Ø)) do
              begin
(6)           remove a dependency g′ with right
                   side RS(g′) from QUEUE;
(7)           if RS(g′) ⊄ DEPEND then
                   begin
(8)                    DEPEND = (DEPEND ∪ RS(g′));
(9)                          Y′ = (Y′ - RS(g′));
(10)          for every dependency g_i in G
                (with left side LS(g_i)) do
(11)             if((LS(g_i) ⊆ DEPEND) AND (g_i ∉ QUEUE))
(13)                 then QUEUE = (QUEUE ∪ g_i);
                   end
              end;
(14)    RETURN Y′;
        end FIND.
          begin (/* main procedure */)
(15)        Y′ = FIND(Y);
(16)        if Y′ = Ø (/* the null set */)
(17)        then PRINT 'YES'
(18)        else PRINT 'NO';
          end.
```

Figure-3.1

$(Y-Y') \in X^+$ with respect to G. So by computing the closure of X we can determine Y' by substracting the closure of X from Y. Hence to compute Y' we have to compute the closure of X i.e. X^+ with respect to G. Let DEPEND be a set variable to hold these attributes i.e. the closure of X. The set DEPEND is initialized to X since by FD1 (the Reflexivity rule) $X \rightarrow X \in G^+$. The set Y' is initialized to Y. While the procedure iterates the values of DEPEND and Y' change repeatedly in such a way that

(i) X->DEPEND is always a consequence of G and

(ii) Y' = (Y-DEPEND) = Y- (the new attributes added to DEPEND in each iteration)

Now to add new attributes to DEPEND, we select an FD, say g' in G whose left side is a subset of DEPEND but the right side is not. By pseudotransitivity rule (FD4) for functional dependencies, the right side of the FD g' is functionally dependent on X and hence can be added to DEPEND and simultaneously the right side of g' (say $RS(g')$) will be substracted from Y', since $X \rightarrow RS(g') \in G^+$. We can continue selecting the FDs of G in this manner, adding and substracting their right sides to DEPEND and from Y' respectively, until no more FDs satisfying this condition. If during any iteration we will find that $(Y-Y')$ is a null set then we can conclude at that point that X->Y is in closure of F and hence it will be unnecessary to iterate further until g all the dependencies of G are checked. The method is formally implemented as Algorithm-1 given in fig.3.1 and the details of an efficient implementation based on an appropriate data structure followed by a proof of correctness are described in the following section.

3.2.1 Proof of termination of Algorithm-1

Both the loop i.e. the outer loop of lines (5)-(13) and the inner loop of lines (10)-(13) are finite loops because G and Y' are finite sets. At each iteration of the outer loop, the inner loop adds a number of FDs to the set QUEUE. But, since the condition, if $(g_i \notin \text{QUEUE})$ then add g to QUEUE in line (11), prevents a dependency to be added more than once to the set QUEUE , hence at most m dependencies can be added to QUEUE. Again since the outer loop is executed at best once for each member of QUEUE the algorithm ultimately reaches to the point that either QUEUE will become empty or Y' become empty (since for each dependency QUEUE during any iteration its right side is substracted from the set Y'), and hence the loop terminates.

3.3 A linear time algorithm for implementation of Algorithm-1 using a simple data structure

The computation of Y' (i.e. lines (6)-(13)) in the algorithm-1 requires an efficient implementation. In this subsection we describe a data structure for a fast on-line execution of the procedure FIND(Y) that runs in $O\|G\|$ time, where $\|G\|$ is the size of the description of the set of FDs G, and the fast algorithm is given in figure 3.2.

In algorithm-2, we assume that the attributes of the set $\{A_1, A_2, \ldots\ldots, A_n\}$ which are appearing on FDs of G, are represented by the numbers 1,2,........,n respectively and also we associate numbers 1,2,.......,m with dependencies $g_1, g_2, \ldots\ldots g_m$ respectively, of G. A linked list LIST(i) for each attribute A_i appearing in G, is constructed where, LIST(i) contains a pointer to each FD that has the attribute A_i on its right side. We also associate a counter COUNTER(j) for each dependency g_j in G where, the counter initially specifies the number of attributes on the left sides of the FDs of G. The linked lists and the counters can be constructed in a single pass over G in $O\|G\|$ time. During the execution of the algorithm COUNTER(j) indicates the number of of the algorithm COUNTER(j) indicates the number of attributes on the left side of the dependency g_j, which are not belonging to the current value of the set DEPEND.

The procedure UPDATE is used to update the counters whenever some attributes are added to DEPEND. When COUNTER(j) associated with the dependency j becomes zero, the left side of the FD j is a subset of DEPEND, hence the FD j is put on the set QUEUE, where QUEUE is the set of all dependencies whose left sides are subsets of the current value of DEPEND.

The algorithm-2 given in figure 3.2 operates essentially as in algorithm-1 by succesfully adding new attributes to DEPEND and substracting the new attributes from Y'. When a set of attributes say R is added to DEPEND in one iteration and which were previously not belonging to DEPEND, then each attribute of R is removed from the left sides of the FDs on which it appears, by calling the procedure UPDATE(R), which updates the COUNTER as well as QUEUE. The algorithm continues until either QUEUE becomes empty or Y' becomes empty.

ALGORITHM-2

Input: A set G of m FDs on attributes $\{A_1, A_2, \ldots, A_n\}$ and an FD $g: X \rightarrow Y$.

Output: "YES" if $g \in G^+$; "NO" if $g \notin G^+$.

Data structures:-

1. Attributes are represented by integers between 1 and n.
2. FDs are represented by integers between 1 and m.
3. LS(j) and RS(j) are arrays of sets containing attributes appearing on left and right sides of the FD j respectively, for each $j \in G$.
4. DEPEND is a set of attributes found to be functionally depending on X so far.
5. Y' is a subset of attributes of the set Y, which are not yet found to be functionally depending on X so far.
6. R is a subset of DEPEND that has not yet been examined.
7. COUNTER[j] is an array containing number of attributes on the left side of each FD j which are not yet found to be in DEPEND.

8. LIST[i] is an array of FDs specifying for each attribute A_i, the FDs with the attribute A_i on their left sides.
9. QUEUE is set of FDs, whose left sides are subsets of DEPEND.

ALGORITHM:-

```
   procedure UPDATE(R):
     begin
(1)      for every attribute A on R  do
(2)       for every dependency j on LIST(i) do
            begin
(3)              COUNTER(j) = (COUNTER(j)-1);
(4)               if COUNTER(j) = 0 then
(5)                              put j on QUEUE;
            end;
       end UPDATE.
     procedure FIND (Y):
     begin
(6)             INITIALIZE: do i = 1 to n
(7)                           LIST(i)=0;
                              end;
(8)                        do j = 1 to m
(9)                            COUNTER(j) = 0;
```

```
(10)                do for each attribute i∈LS[j];
(12)                        LIST[i] = (LIST[i] U {j});
(13)                        COUNTER[j] = (COUNTER[j] +1);
                    end
            end;
(14)            make QUEUE empty
15)         DEPEND = X;
(16)            Y' = Y;
(17)            UPDATE(X);
(18)    while ((QUEUE is not empty) AND (Y is not
            empty)) do
        begin
(19)        remove a dependency i from QUEUE;
(20)        if RS[i]⊄DEPEND then
                begin
(21)                TEMP = DEPEND;
(22)                DEPEND = (DEPEND U RS[i]);
(23)                Y' = (Y' - RS[i]);
(24)                R = (DEPEND - TEMP);
(25)                UPDATE(R);
                end
        end;
(26)    RETURN Y;
    end FIND.
```

```
begin (/* main procedure */)
        Y' = FIND(Y);
          if Y' = Ø (/* the null set */)
           then PRINT 'YES'
            else PRINT 'NO';
      end.
```

Figure-3.2:- A Linear Time Algorithm for the Membership Problem for FDs.

3.4 Analyzing the membership algorithm

To prove the correctness of the angorithm-2 we first examine the initialize step (i.e. lines (6)-(17)). Lines (6) to (13) consists primarily of a scan of G, performing a constant number of operations for each attribute on the left sides of FDs of G, therefore this part terminates (since the number of attributes is finite) and takes time O‖G‖. At the initialize step, the followings hold.

(i) For each g_i in G, COUNTER(i) = |LS(i)|, where |LS(i)| is the length of the left side of the FD g_i.

(ii) For each A_i in the set $\{A_1, A_2, \ldots\ldots A_n\}$, LS(i) contains a list of FDs with A_i on their left sides.

(iii) The set QUEUE is initialized to empty set and the sets DEPEND and Y' are initialized to X and Y respectively, where X and Y are the left side of the given FD g:X->Y, respectively.

The total cost of all the calls to the procedure UPDATE in lines (17) and (25) can be computed as follows.

The cost of executing the lines (1)-(5) once is distributed among the dependencies on LIST(i). Putting a dependency j on QUEUE requires a constant time since only a pointer has to be moved. So a constant time is assigned to each dependency on LIST(i) in one iteration of the loop. During the execution of FIND(Y) each LIST(i) is traversed at most once, and the cumulative cost of each dependency is proportional to the length of its left side. Thus, the total cost of all calls to the procedure UPDATE is no more than $O\|G\|$ time.

The main body of the algorithm is the loop of lines (18)-(25). To prove the termination of this loop, we note that the loop is executed once for each member of QUEUE. Since each dependency is put on QUEUE not more than once at most m dependencies can be added to the set QUEUE. Thus the loop of lines (18)-(25) can execute at most m-times and therefore must terminate in $O\|G\|$ time. Hence the entire algorithm terminates and the running time of the procedure is $O\|G\|$ time. While the worst case time of the algorithm-2 is $O\|G\|$, the running time will be frequently much batter. First, if G contains many FDs whose left sides are disjoint from X^+ (the closure of X with respect to G), then these FDs will never be added to QUEUE and hence QUEUE will become empty much earlier and the number of iteration of the loop on lines (18)-(25) will become very less. Again, if during any iteration in loop ((18)-(25)), it will be found that Y' is empty, the iteration will stop and hence will decrease the running time of the algorithm considerably.

3.5 Application of membership algorithm

The membership algorithm can be applied to solve several FD problems that are related to automatic schema synthesis such as:

(1) To elimanate redundant attributes from a given FD with respect to a set of FDs.

(2) To find various types covers of a set of FDs such as,the non-reduntant cover,the minimal cover and the minimum cover which are required for synthesizing normalized database schemas from a set of FDs.

(3) Also since there exists an equivalence between FDs and propositional formulas such as Horn clauses with at most one negative literal, the linear time algorithm can also be applied to decide if a propositional formula is a tautology.

3.6. Conclusion

One of the objective of this dissertation is to develop an efficient algorithm for the membership problem of the functional dependencies in relational database. The advantages of the proposed algorithm are in distinct contrast to the inadequacies of previous research for the membership problem.There have been a number of methods proposed for this problem over the years. For implementation point of view while the algorithm given by Beeri and Bernstein is considered to be a pioneer one, it has shown here that the algorithm can be improved considerably to reduce the implementation time.

BIBILOGRAPHY

1) AHO, A. V., BEERI, C. AND ULLMAN, J. D. The theory of joins in relational databases. ACM Trans. Database Syst. 4,3(Sept. 1979), pp.297-314.

2) ARMSTRONG, W. W. Dependency structures and database relationships. Information Processing, North Holland, Amsterdam, 1974, pp.580-583.

3) BACHMAN, C. W. Data structure diagrams. Data Base 1,2(1969), pp.4-10.

4) BEERI, C. On the membership problem for functional and multivalued dependencies in relational databases. ACM Trans. Database Syst. 5(1980), pp.141-159.

5) BEERI, C. AND BERNSTEIN, P. A. Computational problem related to design of normal form relational schemas. ACM Trans. Database Syst. 4,1(March 1979), pp.30-59.

6) BEERI, C., FAGIN, R. AND HOWARD, J. H. A complete axiomatization for functional and multivalued dependendencies in database relations. Proc. ACM SIGMOD, (Aug. 1977), pp.47-61.

7) BEERI, C., BERNSTEIN, P. A. AND GOODMAN, N. A sophisticate's introduction to database normalization theory. Proc. 4th Int. Conf. Very Large Data Bases, (Sept. 1978), pp.113-124.

(8) BERNSTEIN, P. A. Synthesizing third normal form relations from functional dependencies. ACM Trans. Database Syst. 1,4 (Dec. 1976), pp.277-298.

(9) BISCUP, J., DAYAL, U., AND BERNSTEIN, P. A. Synthesizing independent database schemas. Proc. ACM SIGMOD Int. Conf. Management of Data, (may 1979), pp.143-151.

(10) BLEIR, R. E. Treating hierarchical data structures in the SDC time shared data management system (TDMS). Proc. ACM Natl. Conf. (1967), pp.41-49.

(11) CARDENAS, A. F. Data Base Management Systems. Allyn and Bacon, Boston, (1979).

(12) CODASYL. CODASYL DATA BASE TASK GROUP Report. Conf. on data Syst. Languages, ACM, New York (1971).

(13) CODD, E. E. A relational model of data for large shared data banks. Commun. ACM 13,6 (June 1970), pp.377-387.

(14) CODD, E. E. Further normalization of the data base relational model. In Data Base Systems, Courant Inst. Computr. Sci. Symp. 6, R. Rustin, Ed., Prentice-Hall, Englewood Cliffs, N. J., 1972, pp.33-64.

(15) DATE, C. J. An introduction to Data Base Systems, 3rd Ed., Addison-Wesly, Reading, MA.

(16) DE BRA, P. AND PAREDAENS, Conditional dependencies for horizantal decompositions. Proc. of 10th Int. Colloq. on Languages Autometa and programming, 1981,

Barcelona. Appeared in Lecture notes in computer Science- Vol. 154, Springer-Verlag,1983, pp.67-82.

(17) DELOBEL, C. AND CASEY, R. G. Decomposition of a database and theory of boolean Switching Functions, IBM J. Res. Develop., 17,5(Sept. 1973), pp.374-386.

(18) FAGIN, R. The decomposition versus the synthetic approach to relational data base design. Proc. 3rd. Int. Conf. Very Large Data Bases, pp.441-446.

(19) FAGIN, R. Multivalued dependencies and a new normal form for relational databases. ACM Trans. on Database Syst. 2, 3(Sept. 1977), pp.262-278.

(20) FAGIN, R. Horn clauses and data dependencies. Proc. 12th ACM Syst. on Theory of Comput., 1980, pp.123-134.

(21) FAGIN, R. Functional dependencies in relational database and propositional logic. IBM J. Res.Devlop. 21,6 (Nov. 1977), pp.534-544.

(22) GALLAIR, H. AND MINKER, J. (Eds). Logic and Data Bases. Plenum Press, 1978, New York.

(23) HAGIHARA, K., ITO, M., TANIGUCHI, K. AND KASAMI, T. Decision problems for multivalued dependencies in relational databases. SIAM J. Comput., 8,2 (May 1979), pp.247-264.

(24) MAIER, D. Minimum covers in relational database model. Proc. 11th Annu. ACM Symp. THeory of Computing, 1979, pp.330-337.

(25) MENDELZON, A. O. On axiomatizing multivalued dependencies in relational database. J. ACM, 26,1 (Jan. 1979), pp.11-14.

(26) NAMBIAR, K. K. Some analytical tools for the design of relational database systems. IEEE, 1980, pp.417-428.

(27) NICOLAS, J. M. First order logic formalization for functional, multivalued and mutual dependencies. Proc. ACM SIGMOD, 1978, 360-367.

(28) OSBORN, S. O. Testing for existence of a covering Boyce-Codd normal form, Inform. Proc. Letters 8,1 (Jan. 1979), 11-14.

(29) RISSANEN, J. Independent components of relations. ACM Trans. Database Syst. 1977, 317-325.

(30) SAGIV, Y., DELOBEL, C., PARKER, D. S. AND FAGIN, R. An equivalence between relational database dependencies and a subclass of propositional logic. J. ACM, 28,3 (July 1981), pp.434-453.

(31) SCIORE, E. A complete axiomatization of full join dependencies. J. ACM, 29,2 (Apr. 1982), 372-393.

(32) ULLMAN, J. D. Principles of Database Systems. Computer Science Press, Rockville, Maryland, 1982.

(33) WANG, C. P. AND WEDEKIND, H. H. Segement Synthesis in logical data base design. IBM J. Res. Develop. 19,1 (Jan. 1975), pp.71-77.

(34) ZANIOLO, C. Analysis and design of relational schemata for database systems. Tech. Rep., Dept. of Comput. Sci., Univ. of California, Los Angles, Calif, July, 1976.

(35) ZANIOLO, C. Mixed transitivity for functional and multivalued dependencies in database relations. Inf. Proc. Letters, 10,1 (Feb. 1980), pp.32-34.

(36) ZANIOLO, C. and MELKANOFF, A. On the design of rel-tional database schemata. ACM Trans. Database Syst. 6,1 (Mar. 1982), pp.1-47.