1870

The Interactive Generation of Functional Dependencies

by

WILLIAM CLEN HUNT

B. G. S., University of Nebraska at Omaha
Omaha, Nebraska, 1972

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

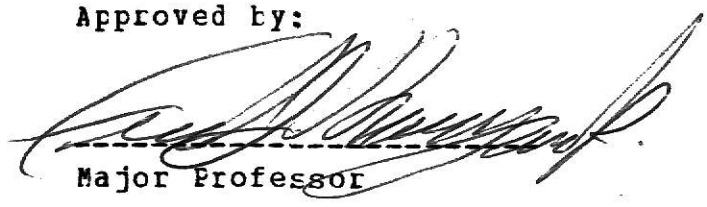
MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1978

Approved by:



Major Professor

Acknowledgement

A special thanks to Dr. Fred J. Maryanski for allowing me to participate in the development and implementation of the system for the automatic generation of third normal form relations, and for his professional attitude during its development. Thanks to Dr. Paul S. Fisher and Edwin W. Basham for serving as members of my committee. The implementation was made possible through a grant from NCR, Inc.

Table of Content

List of Figures

1.0 INTRODUCTION

Information is a prime commodity in today's hurried existence. Managers make decisions based on the best information available to them at the moment. Strategic planning is based on information compiled from many different sources. Every aspect of modern day life is influenced by information. Information is often related to power.

What is information? It is the meaningful interpretation of data which has been collected and intergrated.(1) To meet the needs of today's soceity data must be collected in great quantities. In order for massive amounts of data to be useful to a user, he must have real-time access to it and have a means of synthesizing the available data into meaningful collections so that it might be interpreted into useful information.

Systems have been developed which provide the user with mass storage of data, quick access to the data, and a means of systhesizing the accessed data. The systems, known as Data Base Management Systems (DBMS), allow a user to retrieve, manipulate, and store data without requiring him to know how the data is organized or stored within the storage device.

The user views the data in prespecified logical groups linked together in some organized structure. The more common DBMS structures are known as network and hierarchy. In such views, data may be represented in three ways:(2)

1. The content of the record (e.g., Maryanski's Dept = CS.);
2. The linkage between records (e.g., Maryanski's Dept record occurs in the hierarchy below the College Record for College of Arts and Science.); and
3. The ordering of records (e.g., All of the department records are stored in alphabetical order.).

Usage of such DBMS requires that the user have knowledge of the representation chosen (e.g., FIND NEXT RECORD of College-Dept Set).

The most recent addition to the DBMS family is represented by the Relational Data Model which makes possible the elimination of the representation-dependency from the user's interface. The relational model represents information at the user's interface only by data value. This representation-independence within a DBMS has generated enough interest within the industral community that research is being funded within the area.

Critical to the development of a Relational Data Base is a user knowledgeable of the data-items which will comprise the data base. Data-items are the smallest logical unit of data within a data base. Data-items with the same characteristics are categorized by Data-Name-Item (Attribute). Each Data-Name-Item (DNI) has a domain within which are all data-items described by it. Sets of Data-Name-Items are combined to form relations or groups of

related Data-Name-Items. Each relation must possess one or more Data-Name-Item which can be used as a key to reference the other Data-Name-Items of the group. The association between the key Data-Name-Items and other Data-Name-Items in the group is referred to as functional dependence. The non-key Data-Name-Items are said to be functionally dependent upon the key Data-Name-Items. The set of Data-Name-Items is known as a Functional Dependency and the data base developer must be able to recognize them within his data.

The implementation of a DBMS based upon the Relational Data Model requires the development of a system which encompasses a minimum of three critical areas.(3)

1) A subsystem to accept the unsophisticated user's data and organize it for processing.

2) A subsystem to interact with the user, thus guiding him in the generation of Functional Dependencies.

3) A subsystem to automatically synthesize the generated Functional Dependencies into Third Normal Form (3NF).

This report presents the design, implementation and testing of the interative subsystem, Functional Dependency Generator (FDGEN), which enables the user to generate Functional Dependencies (FD).

2.0 BACKGROUND

A user interfaces with a DBMS through an application program or query language. The program provides the user with predefined views of data, independent of their stored configuration. The life expectancy of application programs is significantly increased when used with relational DBMS.(4)

A relational data base provides for the collecting of data-items in a relation and the formation of logical associations between the relations.

The purpose of a relational data base is fourfold:(5)

1. To free the collection of relations from undesirable insertion, update, and deletion dependencies;
2. To reduce the need for restructuring the collection of relations as new types of data are introduced, and thus increase the life span of application programs;
3. To make it feasible to represent any relation in the data base; and
4. To make the collection of relations neutral to the query statistics, where these statistics are liable to change as time goes by.

Before a Relational Data Base can fulfill its purpose its relations must be in Third Normal Form.

2.1 THE PROBLEM OF THIRD NORMAL FORM

To meet these requirements of a relational data base each data relation must be in Third Normal Form. Each must meet the following test: (6)

1. Each data-item in the relation must be based on a simple domain (it must be atomic);
2. Within each relation data-items are separated into key data-items and non-key data-items. Every non-key data-item must be fully dependent on every key data-item in the relation;
3. Every non-key data-item in the relation must be independent of all other non-key data-items in the relation; and
4. Every key data-item in the relation must be fully dependent on every other key data-item in the relation of which it is a part.

A data relation cannot be tested to see if it is in 3NF before its Functional Dependencies have been identified. A Functional Dependency among the data-items of a relation must be of the following form: (7)

A data-item D of relation C is functionally dependent upon data-item B if, at every instant of time, each B-value in C is associated with only one D-value. The relationship is expressed by the notation B --> D, and says that B determines D or D depends on B. Likewise,

a set of data-items in C may be functionally dependent on another data-item or set of data-items. The data-item (or set of data-items) on the left side of the arrow (B) is called the Determinant.

A user often experiences difficulty trying to determine the Functional Dependencies among the data-items of a data relation, thus increasing the probability that the schema designed would not meet the requirements of a relational data base.

2.2 PROJECT

The automatic generation of 3NF relations is a major step toward the development of relational data bases. NCR, Inc. provided a grant to fund the development of a prototype system which would interactively guide an unsophistocated user in the development of 3NF relations. The system design provided for four subsystems: (8)

1. The User Interface subsystem - An interactive program which reformats input data into hierarchical records, called reports, based on responses to queries from the system and produces an output file of records;
2. The FDGEN subsystem - An interactive program which inputs the data produced by the User Interface subsystem, interactively generates Functional Dependencies and produces as output a file of Functional Dependencies and a hardcopy listing of

Functicnal Dependencies fcr the user;

3. The Functional Dependency Analysier (FDA) subsystem - A prcgram which inputs the data produced by the FDGEN subsystem and synthesis the Functional Dependencies intc 3NF relaticns. It produces as output a hardcopy of the 3NF relaticns; and

4. The Editor subsystem - An interactive program which provides the user with the means to insert, delete, or change any portion of the original input file.

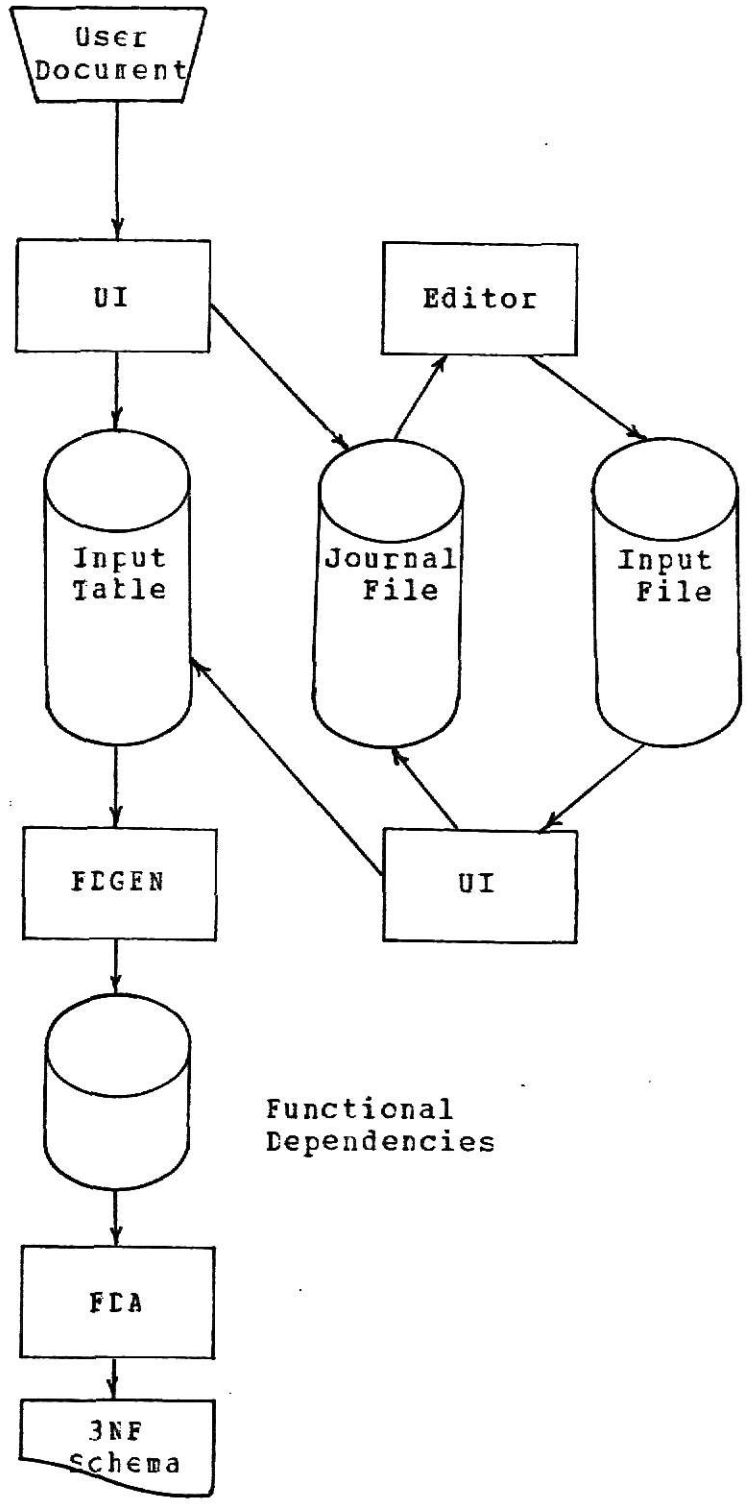
The diagram of the system is pictured in figure 2-1.(9)

2.3 FDGEN

The generation of Functional Dependencies (FD) is not an automatic process. It requires the interaction of a user who is familiar with the data-items input intc the system. Data-items entered as a group must relate to each other so that a logical relaticn can be established. The logically related data-items become the input to the FDGEN. The subsystem defines terms, provides instructicns, and directs the user in the manipulaticn of the related data-items enabling him to form the desired FD. The generated FD's are stcred in an output file fcr later use as input for the FDA.

System Diagram

Figure 2-1



3.0 FUNCTIONAL DEPENDENCY GENERATOR

The Functional Dependency Generator (FDGEN) is an interactive program which accepts as input a file generated by the User Interface Program and produces as output a Functional Dependency Table (F-D-Table) which is stored on disc and used as input for the Functional Dependency Analyzer Program(FDA).(10)

The FDGEN processes the input data an constructs a Data Name Table (DNT) which is used throughout the remainder of the program. The processed data is presented to the interactive user along with the necessary instructions to guide him through the various phases of functional dependency generation. All processed data has associated with it a hierarchical code which allows distinctive data relations to be generated and presented to the user. The user's primary objective is to redefine the generated relations into logical Functional Dependencies(FD). Data manipulation facilities are provided which allow the necessary reconstruction of relations. A hardcopy of the generated FD's is provided to the user upon completion of data manipulation.

An example of a reconstructed relation is illustrated by the following relation with Data-Name-Items (DNI) OWNER, HOUSE, CAR, BUSINESS, and FARM. The user would manipulate the DNI's so that the following FD is generated and output.

OWNER --> HOUSE CAR BUSINESS FARM

The FD says that OWNER identifies the other Data-Name-items

within the relation.

3.1 THE FDGEN ALGORITHM

The algorithm for the generation of FD's is seperated into eleven logical steps.(11)

Step 1. Read the next record from the Input File into a buffer. The format and function of the Input File are provided in figure 3-1.

Step 2. Delete the first four characters (LINE #) from the buffer. Assign the fifth and sixth characters to Status and level respectively. Separate the remaining data in the buffer and assign each delimited set of characters (ATTRIBUTE) to a unique position within the DNT. Assign appropriate values to the associated fields. Link each level '0' position (REPORT NAME) to its successors level '0' position. Mark the end of the DNT. The format and function of the DNT are provided in figure 3-2.

Step 3. When all records have been read and all data has been stored in the DNT display each DNI one at a time. Ask the user to identify all Virtual-Data-Items. Enter a code '3' in the determ-flag field of the DNT for each DNI identified as a Virtual-Data-Item. A Virtual-Data-Item is a DNI which is

Input File Function and Format

Figure 3-1

Line no	Status	Level	Text
XXXX	X	X	X-70-X

Input File Format

Function of the Input File: To store user input text (DNI's) with their associated attributes (the level of the group, its status , and the group's position within the table (line no)).

Data Name Table Function and Format

Figure 3-2

Name	Level-no	Link-code	Determ-flag
X-10-X	X	XXX	X

Data Name Table Format

Function of the Data Name Table: Initially to store in the prescribed format the input data from the Input File as well as the associated special link codes (concatenation links) which are derived from the associated level-no and status. Updated codes will be entered as DNI's are defined through user interaction.

derivable from other DNI's within the data base. The Virtual-Data-Items selected will be eliminated from the report. Figure 4-4 pictures an example where the DNI PRICE has been identified as a Virtual-Data-Item and a code '3' has been entered.

Step 4. Identify potential Determinants by performing an intersection by DNI of each report with every other report in the DNI. For each intersected DNI enter a code of '2' in its associated determ-flag field in the DNT. An example of intersected DNI's is pictured in figure 4-5.

Step 5. Display for the user the list of all DNI's of the current report which have associated with them identical link codes. If a displayed DNI has code '2' in its associated determ-flag field display an asterisk to its right. The asterisk identifies a DNI as a potential Determinant of the displayed group. The user must identify one or more (possibly all) DNI's as Determinants of the group. An example of a marked intersected DNI is pictured with its group on page 63.

Step 6. (a) If the current group from which the Determinant was selected has a level greater than the levels of previously considered groups from the current report, then Determinants for the previously considered groups must be concatenated to the current Determinant to form the Determinant Set for

the current group. The concatenated Determinant Set MODEL, ENG-NC, STYLE, EXTERIOR, and INTERIOR is shown in the Concatenation Stack pictured in figure 4-8.

(b) If the group from which the Determinant was selected has a level greater than one but less than or equal to any level of a previous group from the current report, then those group Determinants whose levels are greater than ~~are~~ or equal to the current group's level must be discarded and the remaining Determinants concatenated to the Determinant of the current group. This situation can best be illustrated by example: Consider the Determinant set in figure 4-8. If the next Determinant selected by the user has associated with it a concatenation link code of '22' all Determinants with codes of '21' would be discarded from the Concatenation Stack and the new Determinant would be concatenated with MODEL to form the new Set.

Step 7. If two or more group Determinants have been concatenated, then display the Determinants and mark the most current with an asterisk. Provide the user with the following options:

a) Elect to leave the determinant configuration unchanged.

b). Elect to redefine all non-current Determinants as non-Determinants.

c). Elect to select one or more Determinants to be redefined as non-determinants.

An example of such a display is pictured on page 67.

Step 8. If two or more group Determinants have been concatenated and they identify themselves, then display the concatenated Determinants and mark with an asterisk those Determinants from the current group. Provide the user with the following options:

a). Elect not to change the determinant configuration.

b). Elect to redefine all non-current Determinants as non-determinants.

c). Elect to select one or more Determinant to be redefined as non-determinants.

The queries which provide the user options are shown on pages 65 and 66. The Determinants are marked the same as those displayed on page 67.

Step 9. Upon selection of an option by the user, add the new FD to the F-D-Table. The FD is added to the table in four steps as follows:

a) Enter the number of Determinants.

Functional Dependency Table Function and Format

Figure 3-3

List
no. of Determinants
Determinant Names
no. of non-determinants
non-determinant names
X-10-X

Functional Dependency Table Format

Function of the Functional Dependency Table: To store the Functional Dependencies generated by the program through interaction with the user. Storage will be sequential from the low index and will begin with a numeric (N), followed by (N) Determinant names, another numeric (M), followed by (M) DNI's. This sequence of events is repeated for each FD to be stored. It is the sole source of input for the FDTABLE output file, and the secondary output of the FDGEN program when written to the line printer.

b) Enter the Determinant names.

c) Enter the number of non-determinants.

d) Enter the non-determinant names.

The F-D-Table format and functions are provided in figure 3-3.

Step 10. Store the F-D-Table on disc for future use as input for the FDA.

Step 11. Restructure the contents of the F-D-Table by FD and output each FD in the format shown in figure 3-4.

OUTPUT FORMAT OF FUNCTIONAL DEPENDENCIES

FIGURE 3-4

DETERMINANT STRING ---> NON-DETERMINANT STRING

The abstract code for the FDGEN algorithm is provided in appendix A.

3.2 AMPLIFICATION OF THE FDGEN ALGORITHM

A detailed explanation of data flow, data manipulation, and user interactions are provided in amplification of the FDGEN Algorithm.

The FDGEN Program accepts as input a current maximum of 200 Data-Name-Items. These DNI's are extracted from the

input file which was produced by the User Interface Program. (12) The file is logically a table of records of four fields. The first field contains a four digit line number which is discarded since it serves no purpose in the FIGEN Program. The second field contains the status of the text field. The status consist of one alpha-character and is used to initiate link code generation. The third field contains the relative level number of the text in the text field. The level number is relative only within the current report. The level number is used to develop a hierarchy among the groups of DNI's within a report. It is analogous to a PL/1 structure or a CCECL record. A picture of a hierarchical structure of DNI's is provided in figure 3-5.

The last field is the text field which contains the DNI's. The data within the text field is delimited by spaces. At least one space must seperate each DNI; however, spaces may also precede the first DNI as well as follow the last one.

The record shown in figure 3-5 is depicted in the format of the Input File and pictured in figure 3-6.

The Input File is read one record at a time until all records have been read. Following the reading of each record the status and level number are extracted and saved for later use, then the DNI's are extracted one at a time from the text field.

Hierarchical Structure of Data-Name-Items

Figure 3-5

```
Record  Automobile
          01 Make
          01 Model
               02 Body-no
               02 Interior
               02 Exterior
               02 Style
               02 Eng-no
                    03 Weight
                    03 Cost
                    03 Price
```

Input File Example

Figure 3-6

LINE	S	L	TEXT
0000		0	Automobile
0010	Y	1	Make Model
0020	Y	2	Body-no Interior Exterior Style Eng-no
0030	Y	3	Weight Cost Price

LINE: Line Number.

S : Status.

L : Level Number.

Text: Data-Name-Items.

The data extracted from the Input File is used to build the DNT, a sample of which is pictured in figure 3-7. The DNT is the major data structure within the program. Data stored in this table is used to service all other structures of the program.

The name field is limited to ten characters; however, it is easily expanded. The level number field may contain any one digit integer while the concatenation link field may contain the characters 'NUL' ~~are~~ or any three digit integer. The determ-flag field is restricted to a one digit integer.

The DNT construction is continuous while data is being read from the Input File and the text field data is being separated. As each DNI is extracted it is placed in the name field of the DNT. If a DNI exceeds the allowable number of characters, an error message will be output and all excess characters will be truncated. The level number for the current record is placed in the level number field. The status does not become a part of the Data Name Table, but is used to initiate code updates.

Each status field may contain one of four characters: 'Y', 'C', 'S', or 'N'. The characters 'S' and 'N' alert the program that the normal hierarchical structuring is about to be interrupted. The character 'Y' simply allows the normal hierarchical structuring to continue while the 'C' causes the program to maintain the current level of structuring.(13) The character 'S' instructs the program to maintain the current level but

Sample Data Name Table

Figure 3-7

Name	Level Number	Concatenation Link	Determinant Flag
Automobile	0	NUL	0
Make	1	11	0
Model	1	11	0
Body-no	2	21	0
Interior	2	21	0
Exterior	2	21	0
Style	2	21	0
Eng-no	2	21	0
Weight	3	31	0
Cost	3	31	0
Price	3	31	3
END		XXX	

that the current record is not a continuation of the prior record. The 'N' causes the program to accept the level number provided by the user , but unless the level number is 1, not to consider the current record as a continuation of any prior records that might possess the same level number. These various conditions control the values that are placed in the concatenation link field.

When a DNI has a level number of 0 and no status, it is assumed to be a report name. The name and level number are entered into the table. The concatenation link field is used to link consecutive reports together. The concatenation link field associated with the current report name will receive the characters 'NUL'. If a predecessor report exists, then the address of the current report is placed in the concatenation link field associated with the predecessor report name. If however, the predecessor report exists in name only (no DNI's associated), an error message will be output and the user will be allowed to delete the erroneous report.

The remaining DNI's associated with the current report are entered in sequence of extraction and the associated concatenation link field is loaded from the Concatenation Code Table which is indexed by the level number. The table is reinitialized at the beginning of each new report. It has nine elements and each element contains a two digit integer. The first element contains 11 and each successive element receives the value of its predecessor plus 10.

For each occurrence of a status, 'S' or 'N', the Concatenation Code Table is updated. The update is equivalent to the movement of a pointer to the next sibling in an iteration traversal of a hierarchical tree structure. The table is updated at the current index (level number) and all succeeding elements by adding 1 to the integer value of each element. To illustrate the table's use, a sample Input File with status 'S' and 'N' and the corresponding DNT are pictured in figure 3-8.

The determ-flag field of each entry is initially set to zero. Following the insertion of the last entry, the end of the table is marked by insertion of an 'END' in the name field and 'XXX' in the concatenation link field. See figure 3-7 for an example of a closed table.

The program provides the user with a tutorial of Virtual-Data-Items and the necessary instruction to allow him to identify Virtual-Data-Items. The only user responses throughout this phase of the program are 'YES' and 'NO'.

The DNI's are read from the DNT one at a time and displayed for the user. Each DNI displayed is also identified by report name. This phase, in essence, provides the user with a sequential view of the DNT.

The user is ask to interact with the program by responding with a 'YES' or 'NO' to the question, is this DNI a Virtual-Data-Item. A 'NO' response allows the program to cycle to the next DNI in sequence;

Sample Extracts of an Input File and Data Name Table with Status of 'S' and 'N'

Figure 3-8

Line	S	L	TEXT
0060	Y	2	text1 text2
0070	Y	3	text3 text4
0080	S	3	text5 text6
0090	N	2	text7 text8

Extract of an Input File

Name	L	C-L	D-F
text1	2	21	0
text2	2	21	0
text3	3	31	0
text4	3	31	0
text5	3	32	0
text6	3	32	0
text7	2	22	0
text8	2	22	0

Extract of a corresponding Data Name Table Segment