Combinatorial and Algebraic Results for Database Relations *

G.O.H. Katona

Mathematical Institute of the Hungarian Academy of Sciences
Budapest, POBox 127, 1364, Hungary

Abstract. A database $R$ has some obvious and less obvious parameters like the number of attributes, the size $|R|$, the maximum size of a domain, the number of some special functional dependencies (e.g. the minimal keys), and so on. The main aim of the paper is to survey some of the results giving connections, inequalities among these parameters. Results of this type give tools to guess the structure of the database having little a priori information. The methods are of combinatorial nature.

1 Introduction

The simplest model of a database is a matrix. The entries in one column are the datas of the same kind (name, date of birth, etc.), the entries of one row are the datas of one individual. Thus, actually, we are dealing with finite sets of homogeneous finite functions which can be illustrated by matrices.

Let us introduce, however, the names of these concepts in the form as they used in the literature. One kind of datas (e.g. name) is called an attribute. It can be identified with a column of the above matrix. The set of attributes will be denoted by $U = \{a_1, \ldots, a_n\}$. The set of possible entries in the $i$th column is the domain of $a_i$. It is denoted by $D(a_i)$. Thus, the data of one individual (row of the matrix) can be viewed as an element $r$ of the direct product $D(a_1) \times D(a_2) \times \ldots \times D(a_n)$. Such an element is called a tuple. Therefore the whole database (or matrix) can be described by the relation $R \subseteq D(a_1) \times D(a_2) \times \ldots \times D(a_n)$, that is, by the set of tuples. If $r = (e_1, e_2, \ldots, e_n) \in R$ then $r(i)$ denotes the $i$ component of $r$, that is, $e_i$.

There might be some logical connections among the datas. For instance, the date of birth determines the age (in a given year). Let $A$ and $B$ be two sets of attributes $(A, B \subseteq U)$. The datas in $A$ might uniquely determine the datas in $B$. Formally we say that $B \subseteq U$ functionally depends on $A \subseteq U$ if

\begin{align*}
\forall i \quad (i \in A) \quad r_1(i) = r_2(i) \quad \text{implies} \quad r_1(i) = r_2(i) \quad \forall i \quad (i \in B),
\end{align*}

where $r_1$ and $r_2$ are elements of $R$.

* The work was supported by the Hungarian National Foundation for Scientific Research, grant number 2575.
It is denoted by $A \rightarrow B$ ([2],[12]). Less formally, $A \rightarrow B$ if any two elements of $R$ having the same values in the attributes belonging to $A$ must have the same values also in $B$. Functional dependencies have a very important role in practical applications, in most of the present paper we will consider them and some natural generalizations.

After this rough introduction of the concepts, let us be more precise. Two levels are distinguished in the database theory. The set $U$ with the sets $D(a_i)$ and the set of logical connections (like functional dependencies) is called the database scheme. It determines the set $R^*$ of possible tuples. The other level is the set $R$ of actual tuples. This is called the instance. The instance obviously has to satisfy the conditions of the database scheme, that is, $R \subseteq R^*$. However, the inclusion is, in general, a proper one.

The traditional investigations of relational databases suppose that the database scheme is a priori given in some way. That is, e.g. some functional dependencies can be deduced either by the logic of the data or by the analysis of (one of) the instances. Then the determination of all functional dependencies is a question of computation. The scheme is fully determined. This information is used then to decompose and store the instances efficiently. Of course, this logical structure of the database scheme can be much more complex, several other (non-functional) dependencies might be known and used.

There is, however a basically different way of considering a database. Knowing some partial (sometimes very week) information on the instances, determine a scheme compatible with the given instances. Or, more modestly, determine some parameters of the scheme. The final goal is to obtain a simple scheme, since it is needed to decompose and store the instances reliably. The more complex the scheme is, the smaller storage space is needed. However it is more probable that the next instance will not be compatible. So actually we should find the simplest scheme determined by the informations on the instances where "simplest" can be defined in different ways. On the other hand, notice that the known information can be of very different nature from the description of the scheme. Since we want to use certain types of connections (which can be strongly used in the decomposition) like, for instance, functional dependencies, but the information could be entirely different from a dependency. Let us call this situation a database relation with unknown structure. (Neither the instance nor the scheme is known fully.)

Examples:

1) The number of attributes and the number of tuples are known. What can be said about the system of functional dependencies? In Section 3 some theorems of the following type are collected. Given the number of attributes and the system of functional dependencies, what is the minimum number of tuples? Obviously, if the given number of tuples is smaller than this minimum then the considered system of functional dependencies can be excluded.

2) The number of attributes and the number of tuples are known. Moreover, there is an assumption on the distribution on each domain. Having no more information it is natural to suppose that the tuples are chosen with random components following the given the distribution. What can be said about the system of functional dependencies? In Section 4 there are some modest results on the expected size of the functional dependencies.