Functional Database Constraints

L. ORMAN
Cornell University, Graduate School of Management, Ithaca, NY 14853, USA

A general notation is developed based on functional languages and Horn clauses to express database constraints. A general notation not only provides a medium for a comprehensive study of all database constraints, as opposed to individual types of constraints, but it also reveals that constraints constitute a major building block of many other database components. In that respect, they are likely to aid in the study of all of those components ranging from queries and transactions to derivations and design algorithms. A universal building block is also likely to be the first step in the development of a unifying theory.

Received October 1986

1. INTRODUCTION

Database constraints are predicates that must be satisfied by the data at all times, for the databases to accurately reflect the reality. Inaccuracies may be the result of a variety of factors ranging from simple clerical mistakes and hardware failures, to fundamental errors in the perception and interpretation of incoming data. In a way, database constraints are gatekeepers of the database system in that they are responsible for insuring the legitimacy and the propriety of all incoming data before they gain entry into the system. This role of insuring integrity is similar to the role of the security system with respect to the users. In fact, a comprehensive system of constraints can be easily extended to serve also as a security system.

There are three major problems with respect to database constraints. They have to be defined, maintained and enforced. Definition requires a language and its associated processors. Maintenance requires a methodology to manipulate constraints, to test their equivalence, redundancy and their consistency with each other. Enforcement requires the generation of procedures that can be run against the database to identify the undesirable data and to prevent their entry into the system. The existing systems admit only the simplest types of constraints, and specialised notation and procedures are associated with each type of constraint. A variety of relational dependencies discussed in literature are typical examples of special constraints, and other classifications of common constraints have appeared in semantic data models. In fact, most discussion of constraints took place in the context of database design within the effort to find an optimum structure for a given body of data. Once in place, the data structure (schema) captures some of the data and only in one direction. Section 2 describes how a relation is necessary to express all constraints uniformly, and because a smaller unit than a relation is necessary to express all constraints uniformly, since many constraints involve only some of the domains of a relation and only in one direction. Section 2 introduces the functional model of data and the

buried in procedures inaccessible to general users. However, the major contribution of a study of database constraints may very well be its aid in understanding other related database facilities such as query languages, transactions procedures and the database design algorithms. Database constraints are usually studied independently of other components of a database system. The major thesis of this article is that not only the constraints constitute a major component of database systems and they are important in their own right, but they are also a primary building block of queries, transactions, procedures and database design algorithms. Consequently, a study of constraints is likely to contribute to all of these topics. A query is characterised by a subset of the database designated for retrieval, and it can be expressed in terms of constraints that divide the universe into those objects that satisfy the constraints (the response set) and those that do not. A procedure can be described as a collection of constraints that must hold for the procedure to be activated, and another set of constraints that must hold after the procedure is executed. These pre and post conditions fully specify the intent of the procedure and this technique is commonly suggested as a declarative specification tool. Database design criteria can also be expressed in terms of constraints. Database design is the process of selecting the best structure for a given collection of data. Two major criteria have appeared in the literature. Minimisation of redundancy corresponds to eliminating data items that are completely specified by constraints; and maximisation of integrity enforcement involves the choice of a structure that minimises the need for explicit integrity constraints by capturing within the structure and automatically enforcing the maximum number of integrity constraints.

The objective of this article is to develop a standard notation to express database constraints, based on a functional notation and Horn Clauses, and to build a methodology to study constraints as a building block of many other components. The functional model of data is used throughout because of its compatibility with the functional notation, and because a smaller unit than a relation is necessary to express all constraints uniformly, since many constraints involve only some of the domains of a relation and only in one direction. Section 2 introduces the functional model of data and the
functional notation. Section 3 introduces a similar notation for database constraints and demonstrates its expressive power. Section 4 established constraints as building blocks of queries, transactions and derivations. Section 5 provides a proof procedure for implied constraints and finally Section 6 suggests design criteria for functional databases based on constraints.

2. FUNCTIONAL MODEL OF DATA

Functional model of data consists of functions defined on data sets. Each function is a set valued mapping from zero, one or more data sets (called arguments) to another set (called range) and characterized by a function name in addition to the names of its arguments and range. Each function identifies a logical relationship between the argument and the range data sets. A data set is a named set of objects. Real world entities in addition to the character strings and numbers used to describe those entities are treated as database objects, and they are grouped into abstractions called data sets.

Example 1.1. A university environment can be modelled using the following functions where a function $F$ from the arguments $A_1, ..., A_k$ to the range $R$ is denoted by $F(A_1, ..., A_k) \rightarrow R$.

- COURSE#(COURSE) \rightarrow STRING
- TEXT(COURSE) \rightarrow INSTRUCTOR
- STUDENT(COURSE) \rightarrow STUDENT
- NAME(INSTRUCTOR) \rightarrow STRING
- PHONE#(INSTRUCTOR) \rightarrow INTEGER
- COURSE(INSTRUCTOR) \rightarrow COURSE
- DEPARTMENT(INSTRUCTOR) \rightarrow DEPARTMENT
- STUDENT(INSTRUCTOR) \rightarrow STUDENT
- TEST(COURSE, INSTRUCTOR) \rightarrow STRING

Each data set of a functional model contains all the objects playing a unique role identified by the name of the set. Consequently, all data sets are required to have unique names. Functions on the other hand may have identical names as long as they are defined on different arguments and hence can be distinguished from context.

A functional notation to express database constraints is naturally compatible with the functional data model. Functional languages provide a number of primitive functions which operate on data and return new data with no side effects. A sequence of functions applied from right to left is called an expression and it constitutes the building block of a purely functional environment.

3. FUNCTIONAL CONSTRAINTS

Constraints are predicates that must hold for all instances of a database. A constraint in a functional environment is composed of a variable, an expression defining the set of permissible values for the variable, and a set of further constraints which define the conditions under which the constraint applies. Using the functional notation, the constraints can be represented as entities with three functions defined on them, each corresponding to one of the three components, where the first two map the constraints to expressions and the third maps constraints to constraints.

Example 3.1. Some simple constraints of the university environment are presented in Fig. 1 using a table format to list all three components of each constraint. Each row corresponds to a constraint and restricts the VARIABLE to take values from VALUE whenever the CONSTRAINTS hold.

The first constraint states that the variable $x$ takes a value from the set INSTRUCTOR, i.e. $x$ is an instructor. A constraint of this sort serve no useful purpose by itself but only as a part of another constraint since all variables are temporary and local. The second constraint restricts the names of instructors to character strings since $n$ takes values from STRING whenever $n$ is contained in NAME(instructor) and instructor is contained in INSTRUCTOR. Similarly, the third constraint restricts the CS instructors to SMITH and JONES. The last two constraints require that all students take all courses, and each student take at least one course respectively. It is interesting to note that in the last constraint no range is specified for the variable course. Consequently, we can only infer the existence of a value that satisfies the constraint. The value is expected to come from the set COURSE since course is a COURSE type variable.

The expressive power of the language is straightforward to demonstrate by equivalence to predicate calculus. Firstly, all predicates of interest can be expressed in terms of sets of objects satisfying that predicate (Russell’s paradox notwithstanding), and the membership in the set of equivalent to the satisfaction of the predicat by a variable or constant. The VALUE function provides the membership operation, and the function application is used to restrict sets through mapping. The rest of the structure consists of a collection of rules where each rule consists of implication of conjunctions. Implication of conjunctions have been studied extensively and have been shown to be equivalent to predicate calculus. Quantifiers are expressed through variables on the left hand side of implications. $x \in X$ on the left-hand side restricts the right hand side to hold for every $x \in X$. 
x on the left-hand side requires the existence of at least one X type value for which the right-hand side holds. Skolem functions\(^\text{10}\) correspond to variables subscripted by other variables, e.g. \(x_y\) means a collection of variables each corresponding to a different value of \(y\). A complete programming language would require a collection of arithmetic and set operations.\(^{9,17}\) These are not necessary for the purposes of this article. Only the negation operation \(\sim\) will be employed to denote all elements not included in a set.

Example 3.2. Some common constraints of commercial databases which are also well known mathematical properties of functions are expressed in Fig. 2, where the first constraint establishes \(\text{NAME(INSTRUCTOR)}\) as a single valued function, the second restricts \(\text{COURSE}(\text{COURSE})\) to be injective, the third states \(\text{PHONE}(\text{INSTRUCTOR})\) as semi-single valued, and the last two restrict \(\text{STUDENT}(\text{COURSE})\) to be full and onto.

In other words, each instructor has exactly one name; a course may have multiple numbers (multiple listings) but the course numbers do not overlap over courses; two instructors may have either exactly the same phone numbers or no common phone numbers at all (office-mates sharing multiple phones); each student takes at least one course; and each course has at least one student.

Example 3.3. Some existence conditions corresponding to Skolem functions\(^\text{10}\) require variables without ranges or variables subscripted by other variables. The first constraint in Fig. 3 states that there is at least one student who takes all courses; the second constraint requires that for every instructor there is at least one student taking all of his courses.

4. CONSTRAINTS AS COMPONENTS

A variety of database constructs such as queries, transactions and derivations can be expressed in terms of constraints. Queries are functions designated for retrieval subject to satisfaction of constraints. Using the functional notation, they can be represented as entities with two functions defined on them:

\[
\text{QUERY} \rightarrow \text{ENTITY} \\
\text{RESPONSE(QUERY)} \rightarrow \text{EXPRESSION} \\
\text{CONSTRAINT(QUERY)} \rightarrow \text{CONSTRAINT}
\]

Example 4.1. Some typical queries of the university environment are listed in Fig. 4. The first query retrieves the names of all instructors; the second lists the names of instructors who teach at least one course; and the third query returns the names of instructors who teach at least one course in every department.

Transactions are procedures that specify insertions, deletions and updates to the database. Each transaction is characterised by two sets of constraints corresponding

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n)</td>
<td>\text{INSTRUCTOR}</td>
</tr>
<tr>
<td>(n)</td>
<td>\text{DEPARTMENT}</td>
</tr>
<tr>
<td>(d)</td>
<td>\text{COURSE}</td>
</tr>
<tr>
<td>\text{student}</td>
<td>\text{STUDENT}</td>
</tr>
<tr>
<td>\text{course}</td>
<td>\text{COURSE}</td>
</tr>
</tbody>
</table>

Figure 1. Constraints of the university environment

| name\(_1\) | \text{NAME(INSTRUCTOR)} |
| name\(_2\) | \text{NAME(INSTRUCTOR)} |
| instructor | \text{INSTRUCTOR} |
| course\(_1\) | \text{COURSE} |
| course\(_2\) | \text{COURSE} |
| p\(_1\) | \text{PHONE} |
| p\(_2\) | \text{PHONE} |
| q\(_1\) | \text{PHONE} |
| i\(_1\) | \text{INSTRUCTOR} |
| i\(_2\) | \text{INSTRUCTOR} |
| \text{student} | \text{STUDENT} |
| \text{course} | \text{COURSE} |

Figure 2. Some common constraints of commercial databases
FUNCTIONAL DATABASE CONSTRAINTS

Figure 3.

<table>
<thead>
<tr>
<th>RESPONSE</th>
<th>CONSTRAINT</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME(x)</td>
<td>x</td>
</tr>
<tr>
<td>NAME(instructor)</td>
<td>instructor</td>
</tr>
<tr>
<td>NAME(instructor)</td>
<td>d</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4. Some typical queries of the university environment

to pre and post conditions. These are conditions that must be satisfied before the transaction is activated and after the transaction is executed respectively. Pre and post conditions fully specify the intent of a transaction, and they are commonly used in the declarative specification of procedures. Using the functional notation transactions are defined as entities with two functions defined on them, both mapping transactions to constraints.

TRANSACTION → ENTITY
PRECONDITION(TRANSACTION) → CONSTRAINT
POSTCONDITION(TRANSACTION) → CONSTRAINT

Example 4.2. Two typical transactions of the university environment are listed in Fig. 5. The first eliminates all the courses taught by ‘SMITH’, and the second inserts the instructor ‘SMITH’ to the database provided that no instructor by that name already exists.

Derivations are used to define some database objects in terms of other database objects. Each derivation is characterized by a set of constraints that completely specify a database object. In a functional environment, each derivation is equivalent to two constraints each declaring one side of an equality as the subset of the other side, where the subset condition is directly expressed in terms of the membership operation.

Example 3.3. The derivation of a function COURSE (STUDENT) can be expressed as follows:

student STUDENT(course) course COURSE(student) course COURSE(student) student STUDENT(course)

The derivation of TEXT(COURSE,INSTRUCTOR) from TEXT(COURSE) and COURSE(INSTRUCTOR) can be expressed as follows assuming that the same textbooks are used in a course by all instructors teaching that course:

t TEXT(c) t TEXT(c,i)
c COURSE(i)
i INSTRUCTOR
t TEXT(c,i) t TEXT(c)
c COURSE(i)
i INSTRUCTOR

5. IMPLIED CONSTRAINTS

Given a collection C of constraints, a constraint c is said to be implied by C if it is satisfied by all databases whenever all the constraints in C are satisfied. The objective in this section is to build an inference system to derive new constraints implied by the existing constraints. The inference system will contain two inference rules corresponding to modus ponens and substitution of free variables, and three sets of axioms reflecting the definitions of basic components.

I. Axioms of variables

(a) x X for every non-empty X.
(b) x y | y x

These two axioms follow directly from the definition of variables. A variable x is assumed to be of type X, i.e. it takes values from the data set X if such a set exists. If a variable x takes its value from a singleton set containing the variable y (brackets omitted when obvious) then the variable y takes the same value as the variable x, i.e. takes a value from a singleton containing only x. This condition is referred to as the equivalence of two variables x and y and they are substitutable for each other.

PRECONDITION

n NAME(instructor) course ‘COURSE(instructor)

n course

i INSTRUCTOR NAME(i) ‘SMITH’

POSTCONDITION

NAME(instructor) ‘SMITH’

COURSE

Figure 5. Two typical transactions of the university environment
II. Axioms of sets
(a) $x \in f(y) \mid x \in f(Y)$
(b) $x \in f(Y) \mid x \in f(y)$

These two axioms follow from the definition of function application. If a variable $x$ takes its value from $f(y)$ then it clearly is also contained in $f(Y)$ since $y$ is in $Y$. The opposite is also true since if $x$ is contained in $f(Y)$ then there is some $y$ in $Y$ such that $x$ is in $f(y)$. It is important to note that in axiom II (b) the variable $y$ appears on the right hand side without appearing on the left hand side. This occurrence of $y$ is said to be a free occurrence.

III. Axiom of impossibility

$x \in P \mid y \in Q$
$x \in \neg P \mid$

The axiom of impossibility states that the impossible premises would imply any conclusion.

Two inference rules are employed to derive new constraints from the existing ones:

I. Modus ponens

Given $x \in X \mid y \in Y$
and $x \in X$
then $y \in Y$

II. Substitution

A variable can be substituted by any other variable of the same type (or typeless), given that all occurrences of that variable are simultaneously substituted and that no free variable is forced to bind. A variable is said to be free in a constraint if it does not occur on the left hand side. A variable is said to be free at a step in an inference if it has not occurred on the left-hand side of any of the constraints used in the inference. An inference is sketched as a series of implications each adding to the known facts by using one of the inference rules, axioms or a given constraint.

Example 5.1. Given that all students take all courses:
student \text{STUDENT} \quad \text{student} \text{STUDENT} (\text{course})
course \text{COURSE}
prove that every student takes at least one course:
student \text{STUDENT} \quad \text{student} \text{STUDENT} (\text{course})
The inference process is given in Fig. 6.

Example 5.2. Given that every student takes at least one course:
student \text{STUDENT} \quad \text{student} \text{STUDENT} (\text{course})
Prove that every student takes all courses!
student \text{STUDENT} \quad \text{student} \text{STUDENT} (\text{course})

The unsuccessful inference process is as follows:
student \text{STUDENT} \quad \text{student} \text{STUDENT} (\text{course},)
course \text{COURSE}
(antecedent) (from premise with substitution of course, for course since course is already bound)

student \text{STUDENT} \quad \text{course} \text{COURSE} \quad \text{student} \text{STUDENT} (\text{course})

(antecedent) (from Axiom I)

Figure 6.

6. DESIGN CRITERIA

The database design problem involves finding the best structure to represent a given body of data. In a functional environment, the problem reduces to deciding which functions to store in the database and which constraints to enforce, leaving other functions and constraints to be implied by the stored ones. Intuitively,
one would like to minimize the redundancy of both the functions and the constraints, and aim for the ideal solution where no function in the database is derivable from other functions, and no constraint is implied by the other constraints in the system. However, these two criteria are not independent and their interaction complicates the process. Moreover, there are often more than one non-redundant solution and they are not always equally desirable. Consider the situation where a function is derivable from others, however some of its constraints are not expressible in terms of new constraints defined on its constituent functions. Under such circumstances, eliminating a redundant function from a database does not achieve all the savings one might expect, since the function has to be reconstructed repeatedly to enforce the constraint.

Example 6.2. Consider the functions DEPARTMENT (STUDENT), DEPARTMENT(ADVISOR) and ADVISOR(STUDENT), all constrained to be single valued. DEPARTMENT(ADVISOR) is derivable from DEPARTMENT(STUDENT) and ADVISOR(STUDENT) since the department of an advisor is the same as the department of his students. However the constraint that enforces DEPARTMENT(ADVISOR) to be single valued cannot be expressed in terms of constraints defined on the individual constituent functions DEPARTMENT(STUDENT) and ADVISOR(STUDENT). This claim follows from the fact that an advisor can have many students, and a constraint forcing all students of an advisor to be in his department necessarily involves both functions and in fact an implicit derivation of DEPARTMENT(ADVISOR).

Example 6.3. Given the functions ADVISOR(STUDENT) and DEPARTMENT(STUDENT) where a student can have only one department, but possibly many advisors as long as they are all in the same department, i.e. the department of the student. DEPARTMENT(STUDENT) is derivable from DEPARTMENT(ADVISOR) and ADVISOR(STUDENT) since the department of a student is the same as the department of his advisors. However, the constraint that enforces DEPARTMENT(STUDENT) to be single valued cannot be expressed in terms of constraints defined on the individual constituent functions DEPARTMENT(ADVISOR) and ADVISOR(STUDENT). This claim follows from the fact that ADVISOR(STUDENT) is not single valued, and the constraint forcing all advisors of a student to the same department necessarily involves both DEPARTMENT(ADVISOR) and ADVISOR(STUDENT), and in fact an implicit derivation of DEPARTMENT(STUDENT).

The relational theory has emphasized the non-redundancy of data and design algorithms based on this objective have been proposed. However the issue of choosing among the many non-redundant schemas representing the same body of data remains an unresolved problem and the study of the relationship between redundancy and constraints remains restricted to certain classes of constraints. A general design criteria would have to encompass not only the non-redundancy of data but also a variety of constraints and their interaction with the non-redundancy criteria. Within the functional notation, since each derivation is a pair of constraints, it is possible to combine these objectives into one of minimising constraints. A minimal set of constraints is a set of constraints where no constraint is implied by the others. Once all redundant constraints are eliminated, and a minimal set is found, then the storage problem reduces to mere selection of those functions essential to the enforcement of this minimal set of constraints. This criterion improves upon the simple non-redundancy criterion by further restricting the choice to one of the many non-redundant models.

Given that DEPARTMENT(STUDENT) is derivable from the others:

\[
\begin{align*}
d & \quad \text{DEPARTMENT(a)} \\
a & \quad \text{ADVISOR(s)} \\
s & \quad \text{STUDENT}
\end{align*}
\]

and that DEPARTMENT(ADVISOR) is derivable from the others:

\[
\begin{align*}
d & \quad \text{DEPARTMENT(s)} \\
a & \quad \text{ADVISOR(s)} \\
s & \quad \text{STUDENT}
\end{align*}
\]

and that all three functions are single valued:

\[
\begin{align*}
d_1 & \quad \text{DEPARTMENT(s)} \\
d_2 & \quad \text{DEPARTMENT(s)} \\
d_3 & \quad \text{DEPARTMENT(s)} \\
a_1 & \quad \text{ADVISOR(s)} \\
a_2 & \quad \text{ADVISOR(s)}
\end{align*}
\]

Figure 7.
The following inference shows that single valuedness of \( \text{DEPARTMENT(STUDENT)} \) follows from the premises 1, 3.2 and 3.3:

\[
\begin{align*}
&d_1 \text{ DEPARTMENT}(s) & d_1 \text{ DEPARTMENT}(a) & a_1 & a_2 \\
&d_2 \text{ DEPARTMENT}(s) & d_2 \text{ DEPARTMENT}(a) & a_1 & a_2 \\
&\text{(antecedent)} & \text{(antecedent)} & \text{(from premise 1)} & \text{(from premise 3.3)} \\

d_1 \text{ DEPARTMENT}(a) & d_1 \text{ DEPARTMENT}(s) & a_1 & a_2 \\
&\text{(from premise 3)} & \text{(from premise 1)} \\
&\text{(from substitution rule)} & \text{(from premise 2)} \\
\end{align*}
\]

The inference described by Fig. 8 shows that the single valuedness of \( \text{DEPARTMENT(ADVISOR)} \) is not implied by the other constraints.

These inferences lead to the conclusion that the constraints 1, 3.2 and 3.3 constitute a minimal set, and the only functions necessary to enforce these three constraints are \( \text{DEPARTMENT(ADVISOR)} \) and \( \text{ADVISOR(STUDENT)} \), since \( \text{DEPARTMENT(STUDENT)} \) is completely defined by the minimal set.

**Example 6.4.** Given the functions of Example 6.2 where both \( \text{DEPARTMENT(ADVISOR)} \) and \( \text{DEPARTMENT(STUDENT)} \) are derivable from the other two, the elimination of either function would result in a non-redundant model. However, the two non-redundant models are not equivalent since the elimination of \( \text{DEPARTMENT(ADVISOR)} \) makes it impossible to enforce some constraints while the elimination of \( \text{DEPARTMENT(STUDENT)} \) does not. The following inferences demonstrate this fact resulting in elimination of the \( \text{DEPARTMENT(STUDENT)} \) in the optimal model:

Given that \( \text{DEPARTMENT(STUDENT)} \) is derivable from the other functions:

\[
\begin{align*}
&d \text{ DEPARTMENT}(a) & d \text{ DEPARTMENT}(s) & a \text{ ADVISOR}(s) \\
&s \text{ STUDENT} & & \text{ (premise 1)} \\
&d \text{ DEPARTMENT}(s) & d \text{ DEPARTMENT}(a) & a \text{ ADVISOR}(s) \\
&s \text{ STUDENT} & & \text{ (premise 2)} \\
\end{align*}
\]

and also given that \( \text{DEPARTMENT(ADVISOR)} \) is derivable from others:

\[
\begin{align*}
&d \text{ DEPARTMENT}(a) & d \text{ DEPARTMENT}(s) & a \text{ ADVISOR}(s) \\
&s \text{ STUDENT} & & \text{ (premise 2)} \\
\end{align*}
\]

\[
\begin{align*}
&\text{(premise 1)} & \text{(from premise 4)} & \text{(from premise 3)} \\
&\text{(by substitution)} & \text{(from premise 1)} \\
\end{align*}
\]

\[
\begin{align*}
&d \text{ DEPARTMENT}(a) & a \text{ ADVISOR}(s) & s \text{ STUDENT} \\
&\text{(premise 1)} & \text{(from premise 4)} & \text{(from Axiom 1)} \\
&d \text{ DEPARTMENT}(s) & d \text{ DEPARTMENT}(a) & a \text{ ADVISOR}(s) \\
&\text{(from premise 2)} & \text{ (from premise 4)} \\
&\text{(from premise 3)} & \text{ (from premise 3)} \\
&\text{(no other inferences)} \\
\end{align*}
\]
and that DEPARTMENT(STUDENT) and DEPARTMENT(ADVISOR) are single valued:

\[ \text{dj DEPARTMENT(s) d, d 2 DEPARTMENT(s)} \]
\[ \text{dt DEPARTMENT(a) d, d 2 DEPARTMENT(a)} \]

(premise 3)

and that all advisors have students and departments:

\[ \text{a ADVISOR(s) d DEPARTMENT(a)} \]
\[ \text{s STUDENT d DEPARTMENT(a)} \]
\[ \text{a ADVISOR d DEPARTMENT(a)} \]

(premise 4)

The inferences in Fig. 9(a) show that the premise 2 is implied by the premises 1, 3 and 4.
The inferences in Fig. 9(b) show that the premise 1 is not implied by the premises 2, 3, and 4.

Since the premise 2 is implied by 1, 3 and 4 and no other inferences hold, \((1, 3, 4)\) is a minimal set of constraints and DEPARTMENT(ADVISOR) and ADVISOR(STUDENT) are sufficient to enforce all constraints, leading to the elimination of DEPARTMENT(STUDENT), which is completely defined by the minimal set.

7. CONCLUSIONS

Database constraints constitute a major component of database management systems. Classifying constraints, and using specialised notations to study each type, prevent the development of a comprehensive theory of constraints. A standard notation based on a functional language and Horn clauses not only provides a medium to study the constraints in more general terms, but it also reveals that the constraints are a major building block of many other database components. In that respect, they may aid in the study of all of those components, including queries, transactions, derivations and design algorithms. This article has provided a general notation to express constraints, and established constraints as a major building block. Future research will be directed at a unifying theory of databases based on constraints. In particular, the completeness of the language has to be proved, extensions have to be made to serve as a programming language, the axiomatic system has to be proved complete and algorithms should be developed to automate the inference process, the design criteria has to be compared against other criteria and algorithms should be developed to automate database design.

REFERENCES