UPDATE SEMANTICS FOR INCOMPLETE DATABASES

Serge Abiteboul **Gösta Grahne**(*)

Institut National de Recherche en Informatique et Automatique
78153 Le Chesnay, CEDEX
FRANCE

ABSTRACT

A database containing some incomplete information is viewed as a set of possible states of the real world. The semantics of updates is given based on simple set operations on the set of states. Some basic results concerning the capabilities of known models of incomplete databases to handle updates are exhibited.

INTRODUCTION

In many database applications, the knowledge of the real world modeled by the database is incomplete. A lot of research has been devoted to the problem of querying these so-called incomplete databases [C, B, IL1, R2, Vr, Vs]. However, a problem at least as delicate has been little studied : The problem of updating them. In this paper, we present a general framework for studying updates of incomplete databases. We then use this framework to obtain results on updates in the context of three known models for incomplete databases.

(*) The work of this author was partially supported by the Academy of Finland.

Two fundamental aspects of incomplete database updates can be distinguished :

- The database contains some incomplete information (e.g. null values) which may be affected by the update. This aspect is studied for instance in [FUV, KUV].

- The update itself is not completely specified. This is the case for view updates [BS], or updates in semantic database models [AH].

Although these two aspects have rarely been considered together, they are certainly intimately tied. If all updates are completely specified, how did some incomplete information arrive in the database in the first place? Also, if the system is not able to interprete a view update, or to (uniquely) propagate the update in a semantic database model, an alternative to the refusal of the update is certainly the acceptance of some incomplete information in the database.

For these various reasons, it is fundamental to understand the relationship between incomplete information, and updates.

In our framework, an incomplete database is simply viewed as a set of complete instances, i.e., the set of all possible states of the real world. An update is then seen as a mapping from sets of instances to sets of instances. We present some large classes of updates based on classical operations on sets. Some of them (e.g., deletion and insertion) are generalizations of classical updates. Others are more tightly related to the incompleteness of the information. In particular, we introduce the concept of subjection which is a novel usage of the classical notion of dependency in the context of updates.

The second major theme of the paper is a validation of these various concepts by applying them to three precise models of incomplete databases. The three models that are

considered here are Codd tables [C, B], naive tables, and conditional tables [IL1]. We introduce updates for these various tables. We then present some important results which highlight the abilities of these tables to support update operations. We believe that this demonstrates that the understanding of the update problem is a necessary step towards understanding the notion of incomplete information at a fundamental level.

The paper is organized as follows. The first section briefly presents some well-known concepts of relational databases. In the second section, the notion of incomplete information is studied at an abtract level. In particular, a classification of updates based on classical set operations is introduced. The concept of representation system of [IL1] is then extended to serve as a basis for the "implementation" of these abstract notions. In the third section, we move to a more concrete ground by presenting three particular known models for representing incomplete information. We then define updates for these three models. The third section also contains the results of the paper. The update capabilities of representation systems based on the three models are exhibited. We also give some negative results showing the limitations of two of these models.

I. PRELIMINARIES

We assume that the reader is familiar with the relational model to the extent of [U].

The following notation will be used throughout the paper. We assume that the database consists of one relation scheme, which is a fixed finite set R of attributes. (All results are though immediately generalizable to multirelational schemata.) We use A, B, .. to denote attributes. With each attribute A in R, there is an associated set called the domain of A, denoted dom(A). The domains are assumed to be (countably) infinite. Each element of a domain is called a constant. A tuple t over R is a mapping over R such that t(A) is in dom(A) for each A in R. For a subset X of R, the restriction of a tuple t to X is denoted t[X]. To precise the value of a particular tuple, we shall use the classical notation <a, b, c, ...>. Here a is the value for the first attribute, b of the second, etc.. . (Some understood ordering of the attributes is assumed.) An instance (or relation) I over R is a finite set of tuples. Instances will sometimes also be called states.

For querying the database, we shall use the relational algebra obtained using the following operations : projection, selection, union, (natural) join, difference and renaming. It is known that this collection of operators form a complete relational algebra, assuming that there is an infinite collection of alternative attribute names A_1, A_2, ... available for each attribute A. The formula used to specify a selection is a boolean expression formed using atoms of the form A=a, where A is in R, and a is in dom(A). A selection will be called positive if its formula does not contain negation.

We will sometimes use the symbol Ω to denote a subset of these operations (e.g., Ω = {projection, join}). By an Ω-expression, we then mean a well-formed expression involving only operators in Ω. For instance, if R = ABC the expression "$(\pi_{AB}(R)) \bowtie (\pi_{BC}(R))$" is an Ω-expression for any Ω containing projection and join. This notion of an Ω-expression will be extended is the natural way to updates. For example, "(delete ... from R) ∪ (insert ... into R)" illustrates an Ω-expression for Ω containing at least deletion, union and insertion.

The existence of an infinite set of variables is also assumed. The variables will mainly be used to express null-values.

In most commercial database systems, three types of updates are considered, namely insertion, deletion and modification. (We use here the model of updates introduced in [AV1]. A more detailed presentation of these operations, together with studies on the equivalence between compositions of updates ("transactions"), and on optimization of such transactions can be found there. The interaction between updates and dependencies is studied in [AV2].) To formally define the update operations, we need the auxiliary concept of "condition".

An elementary condition over R is an expression of the form A=a or A≠a, where A is in R and a is in dom(A). A condition is a conjunction of elementary conditions. (A condition is thus a syntactically restricted form of a selection formula). A tuple t satisfies a condition F if the condition evaluates to true when each attribute A appearing in F is substituted by t[A]. In the following, we consider only meaningful conditions, that is, conditions that can be satisfied by some tuple. We now have the following definitions.

An insertion on R is an expression ins(F), where F is a condition specifying a complete tuple t over R (i.e. F lists values for all attributes in R). The result of performing an insertion on an instance I is defined by : ins(F)(I) = I ∪ t.

A deletion on R is an expression del(F), where F is a condition over R. The result of performing a deletion on an instance I is

defined by : $del(F)(I) = I - \{t \mid t \text{ satisfies } F\}$.

A modification on R is an expression mod(F;F'), where F and F' are conditions, the latter one not containing inequalities. The result of a modification on an instance I is such that the condition F selects the tuples to be updated, and F' lists new values for some of the attributes.

For instance, if R = {FIRSTNAME, NAME, ADDRESS}, and I is an instance over R, then mod(FIRSTNAME = toto; ADDRESS = Paris)(I) changes in I the address of all persons with firstname toto to Paris.

For incomplete databases, we shall define some additional update operations. They will be considered in the next section.

To conclude the preliminaries, we present some notation for operations on sets of instances. Let X and Y be sets of instances over the same scheme R.

Union : $X \cup Y = \{I \mid I \text{ in } X \text{ or } I \text{ in } Y\}$.

Intersection : $X \cap Y = \{I \mid I \text{ in } X \text{ and in } Y\}$.

Difference : $X - Y = \{I \mid I \text{ in } X \text{ and } I \text{ not in } Y\}$.

Pairwise union : $X\ (\cup)\ Y = \{I \cup I' \mid I \text{ in } X \text{ and } I' \text{ in } Y\}$.

Pairwise intersection : $X\ (\cap)\ Y = \{I \cap I' \mid I \text{ in } X \text{ and } I' \text{ in } Y\}$.

Pairwise difference : $X\ (-)\ Y = \{I - I' \mid I \text{ in } X \text{ and } I' \text{ in } Y\}$.

II. ON INCOMPLETE INFORMATION

In this section incomplete information is studied at an abstract level. The meaning of an incomplete database is expressed in set theoretic terms. Updates are then defined as set theoretic operations. Finally, the notion of representation system introduced by Imielinski and Lipski [IL1] is extended to handle updates. As for queries, or in the context of transactions, it is stressed that the system should be able to handle compositions of operators (i.e. expressions) rather than just single ones. By this, we mean that the application of one update, and then another one, should yield the same result as a single step composition of the two updates.

II.1 Semantics.

Databases containing incomplete information are usually stored as relations with some sort of null-values in the tuples. In Figure II.1, we see a very simple example of such a relation. (The symbol @ represents a null-value.)

NAME	ACTIVITY
toto	siesta
lulu	@
@	music

Figure II.1

The intended meaning of this kindergarten database is that toto is taking part in the siesta activity, lulu is involved in some activity, but we do not know which one, and there is someone with an unknown name playing music. This intended meaning has to be expressed in some formal way by assigning some semantic interpretation to the database. The interpretation that we are going to use is that a possible state of the real world according to this database can be obtained by substituting each null-value in the relation by a value from the proper domain. Thus, if we assume that the domain of the attribute NAME is {toto, lulu, zaza}, and that the domain of ACTIVITY is {siesta, music}, then all the possible states are given in Figure II.2. (To save space, we abbreviate NAME and ACTIVITY by N and A, and use t, l, z, s, and m for the various names and activities.)

N A	N A	N A	N A	N A	N A
t s	t s	t s	t s	t s	t s
l s	l s	l s	l m	l m	l m
t m	l m	z m	t m		z m

Database X

Figure II.2

For the rest of this section, we will consider a database with incomplete information simply as a set of instances one of which corresponds to the true state of the real world. If this set contains only one element, we have an ordinary (complete) database. Now the semantics of an operation on a database X, i.e., a set X of instances, can be defined from the operation and the set X. For instance, consider a query q. The result of this query q on the database X is given by $q(X) = \{q(I) \mid I \text{ in } X\}$. So, for X as in Figure II.2, if we ask for the activities of toto, by making a selection with the formula N=toto, the answer will be the set of states in Figure II.3.

N A	N A
t s	t s
t m	

q(X)

Figure II.3

The obvious meaning of this answer is that either toto is involved in siesta and music, or just in the siesta activity. From this, we know for sure that toto is involved in the siesta activity. This "for sure" knowledge can be obtained as $\bigcap_{I \text{ in } X} q(I)$, i.e. the set of tuples which appear in all possible states of the answer.

The simple foundation of the above semantics is the following : The database contains all the possible states of the real world, and by applying the query on all possible states, we will get all possible answers. We adopt the same simple strategy for all update operations. For instance, if we want to insert a tuple into an incomplete database, this tuple should be inserted into the true state of the real world. Since we only know the set of possible states, we insert the tuple into each possible state, thus obtaining all possible results of the operation.

The situation is not always as simple since the update itself can be incompletely specified. Suppose that we want to insert the fact that zaza is involved in some unknown activity. This fact can be expressed as a set of one-tuple instances {{<zaza, siesta>}, {<zaza, music>}}, i.e. as an incomplete database Y_1. Inserting this fact about zaza into a database X comes to "inserting" Y_1 into X. One possibility of the result is obtained by the insertion of a possibility of Y_1 into a possibilty of X. Thus the result is accomplished by the pairwise union X ($\cup$) Y_1. We shall call this operation (general) insertion. If X is the database in Figure II.2, and Y_1 as above, the result X ($\cup$) Y_1 of the insertion of Y_1 in X is given in Figure II.4.

The use of a classical set operation to define a class of updates suggests that, more generally, updates can be viewed as set theoretic operations. Indeed, we now turn to other classes of updates which will correspond to the set operations defined in the preliminaries. The other classes of updates are : deletion, integration, subjection, negative subjection and augmentation.

N A	N A	N A	N A	N A	N A	N A	N A
t s	t s	t s	t s	t s	t s	t s	t s
l s	l s	l s	l s	l s	l s	l m	l m
t m	t m	l m	l m	z m	z m	t m	t m
z s	z m	z s	z m	z s		z s	z m

N A	N A	N A	N A
t s	t s	t s	t s
l m	l m	l m	l m
z s	z m	z m	z m
		z s	

X ($\cup$) Y_1

Figure II.4

The first one, namely deletion, is based on pairwise difference. If a tuple is to be deleted from a set of possible states, this is accomplished by deleting it from every possible state. A set of possible states Y can be deleted from another set of possible states X by the set theoretic operation of pairwise difference X (-) Y. This is the definition of (general) deletion. If we now want to delete all facts about toto from the database X in Figure II.2, this is obtained by taking X (-) Y_2, where Y_2 is the database {{<toto, siesta>, <toto, music>}}. (Note that Y_2 contains one instance with two tuples). The result is shown in Figure II.5.

N A	N A	N A	N A	N A
l s	l s	l s	l m	l m
	l m	z m		z m

X (-) Y_2

Figure II.5

The knowledge contained in two databases X and Y can also be integrated into one database by taking the tuples common to a possible state in X and a possible state in Y as one possible state of the new database. This integration update is defined by the pairwise intersection X ($\cap$) Y.

There are still three other set theoretic operations that we are going to use for defining updates. The first one of these is intersection. Suppose that we have a database X, and that our knowledge of the real world has increased so that we know that only some specific states are possible. This knowledge can for instance be that toto is involved in only one activity. Now if we take Y_3 as the set

of all instances where toto is involved in at most one activity, this knowledge can be put into a database X by keeping only those instances in X that also are in Y_3, i.e. by replacing X with $X \cap Y_3$. We have thereby defined the subjection update. The name subjection means that the set of possible states are subjected to be among the states in Y_3. To illustrate this operation, let Y_3 be as just defined (e.g., the set of instances where toto is involved in at most one activity), and X the database in Figure II.2. Then the subjection of X by Y_3 (i.e., $X \cap Y_3$) is shown in Figure II.6.

N	A
t	s
l	s
l	m

N	A
t	s
l	s
z	m

N	A
t	s
l	m

N	A
t	s
l	m
z	m

$X \cap Y_3$

Figure II.6

In a similar fashion, we can incorporate negative knowledge into a database. Suppose that we know that lulu is not involved in the music activity. Let Y_4 be the set of all instances where lulu is involved in music. Our negative knowledge is then incorporated in the database by replacing X with $X - Y_4$, i.e. by keeping those states in X that are not in Y_4. (Note that we are making here the closed world assumption [R1] by assuming that the absence of a fact in a possible state means that the fact is false in that state.) We shall call this kind of operations negative subjection.

The reader has certainly noted the resemblance between a subjection update and dependency enforcement. Indeed, we are going to use dependencies to define certain subjection updates. For instance, if we know that all persons are taking part in at most one activity, the set Y used for the subjection can then be the set of all instances satisfying the functional dependency NAME -> ACTIVITY. The main difference between subjection update, and dependency enforcement is that dependencies are usually regarded as time independent constraints that the database has to always obey, whereas the constraint of a subjection update has to be satisfyed only immediately after the update is applied.

It may be worthwhile at this point to emphasize the difference between the updates based on pairwise operations and the subjection updates. Suppose that X is updated with Y. In the pairwise operations, we are asked to change the content of the database based on the information in Y. Indeed, the true state of the database may be altered. Conversely, in a subjection, the information in Y is only used to restrict the set of possible states of the database. The true state of the database is not modified. It is only our knowledge of it which is getting more precise.

The sixth update is called augmentation. The augmentation of a database X with a database Y is defined as $X \cup Y$. The obvious intuitive meaning is that we knew (with X) a set of possible states, and we now know that other states (Y) are also possible. Hence the new set of possible states are those that are either in X or in Y.

To conclude this presentation of general update operations, we now present a last class of updates which is a generalization of the modification operation on complete databases. If we want to "modify" an incomplete database, the modification has to be done on every possible state. Formally, a modification is a function f from instances to instances. When applied to a database X, a modification transforms X into a database f(X) defined by : $f(X) = \{f(I) \mid I \text{ in } X\}$. Consider the database X' in Figure II.5, and the modification f= mod(N=zaza;A=siesta). The result f(X') is shown in Figure II.7.

N	A
l	s

N	A
l	s
l	m

N	A
l	s
z	s

N	A
l	m

N	A
l	m
z	s

f(X')

Figure II.7

We note that like the other updates, the modification can be incompletely specified. Thus, if we want to change either zaza's or lulu's activity to siesta, this is accomplished by the modification F, where F is a set of two modification functions. The first one is as in the previous example, and the second one is modifying the activity of lulu into siesta. The result of the modification in this case is defined as $F(X) = \{f(I) \mid f \text{ in } F \text{ and } I \text{ in } X\}$. If X' is the database in Figure II.5, the result is given in Figure II.8.

N	A
l	s

N	A
l	s
l	m

N	A
l	s
z	s

N	A
l	m

N	A
l	m
z	s

N	A
l	s
z	m

F(X')

Figure II.8

II.2 Representation Systems

In this section, we briefly present the notions of strong and weak representation systems [IL1], generalizing them to handle updates.

So far we have only assumed that an incomplete information database is a set of complete instances, and that one of these instances corresponds to the true state of the real world. As soon as we move towards an implementation of the database we are faced with the problem of representing the database in some compacted fashion, especially since we want to be able to deal with infinite sets of possible states. Representation techniques for incomplete databases are often based on tables, which are essentially relations with null-values of some sort.

This leads to the notion of representation. A representation is a pair <S,rep>, where S is a set of tables, and rep is a mapping from S to sets of complete instances. Intuitively S is the set of all tables of a particular choice, rep is the mapping that gives the set of states represented by a table.

Once a way of representing the database is chosen, we have to assure that we also are able to represent the result of an operation (update or query) in the same way. Furthermore, since the database is available only in the form of its representative, we must have an algorithm based on the representative to compute the result of the operations.

We are thus interested in systems composed of a representation <S,rep>, and a set Ω of allowed operations (e.g. join and insertion). To merit the name representation system such a system should have some particular properties. We are going to distinguish between "strong" and "weak" representation systems. We first present the strong version.

Definition : A triple <S,rep,Ω> is a strong representation system iff for each table T in S and each Ω-expression f, there exists a table T' in S such that

1) rep(T') = f(rep(T))
2) There is an algorithm to compute T' from T and f.

This definition is in a sense the strongest requirement that can be made for a representation system, since it states that it is possible to compute and represent the exact result of all allowed expressions. As noted in [IL1], this requirement is too strong for some kinds of tables even with Ω containing only a subset of relational algebra. That is, the result of some expressions are not representable by any table of the particular kind.

When the system is too weak to represent results of arbitrary Ω-expressions, we may accept approximations of those results : Suppose that instead being given the answer to a query as $\{f(I) \mid I \text{ in } X\}$, the user is given $\bigcap_{I \text{ in } X} f(I)$, i.e. only the information that is true in all possible states of the real world. Now, suppose further that this user has a query language based on some Ω at his disposal, and that for two databases X and Y and all Ω-expressions f, $\bigcap_{I \text{ in } X} f(I) = \bigcap_{I \text{ in } Y} f(I)$. In this case, the user cannot distinguish between X and Y. Thus, X and Y are, in some sense, equivalent (since they give the same answer to all queries). This equivalence is called Ω-equivalence and denoted $X \equiv_{\Omega} Y$. (For a more detailed discussion on Ω-equivalence, see [IL1].) The Ω-equivalence motivates the following notion of weak representation systems.

Definition : A triple <S,rep,Ω> is a weak representation system iff for each table T in S and each Ω-expression f, there exists a table T' in S such that

1) $rep(T') \equiv_{\Omega} f(rep(T))$, and
2) There is an algorithm to compute T' from T and f.

Note that a strong representation systems is obviously also a weak representation system.

In the next section, we are going to present three particular representation systems, based on three different kinds of tables.

III. UPDATES AND REPRESENTATION SYSTEMS

In this section, we first briefly present three fundamental models introduced for dealing with incomplete information, namely Codd, naive and conditional tables. We also briefly review well-known results concerning queries on these tables. We then study the problem of updating these tables. We present a variety of results which highlight the possibilities and limitations of each type of table.

III.1 Tables

The first system that we are going to consider is based on the notion of Codd table [C,B]. An example of Codd table is given in Figure III.1. As we can see, a Codd table is a relation containing some particular symbols '@' indicating unknown values. A possible complete state specified by a Codd table is obtained by replacing each symbol '@' by a value of the corresponding domain. Given a Codd table $T_@$, the set of all possible states is denoted $rep(T_@)$.

SUPPLIER	ADDRESS	PRODUCT
@	Romorantin	nails
toto	@	bolts
zaza	@	nuts

Codd table $T_@$

Figure III.1

The second system is based on the notion of naive table [IL1]. An example of a naive table is given in Figure III.2. A naive table is a relation containing constants (e.g., nails, zaza, ...), and variables (x, y, ...). The variables are used to indicate unknown values. Moreover, since a variable can appear more than once in the table, one has the possibility to represent knowledge like: "the addresses of toto and zaza are unknown, but we know that they have the same address". A possible state is obtained by first assigning a value to each variable, then replacing each variable by its assigned value. Given a naive table T_N, the set of all possible states is denoted $rep(T_N)$.

SUPPLIER	ADDRESS	PRODUCT
x	Romorantin	nails
toto	y	bolts
zaza	y	nuts

Naive table T

Figure III.2

The last system is based on the notion of conditional table (C-table) [IL1]. An example of a conditional table is given in Figure III.3. Like in naive tables, we have constants and variables. The table has also a global condition, and each tuple has a local condition. A global or local condition is any boolean combination of elementary conditions. In the example, "y ≠ Romorantin" is the global condition, and "x ≠ mimi", "true" are local conditions. To obtain a possible state, one must first have an assignment of the variables which make the global condition true, then in each tuple such that the local condition is true (for that assignment) replace the variables by their associated values. (Tuples with false local condition are just discarded). Intuitively, a C-table allows us to also represent information like: "If x is not mimi, then x is supplying nails". Note that if naive tables allow to state equalities between unknown values, C-tables allow also to state inequalities. Given a C-table T_C, the set of possible states is denoted $rep(T_C)$. For T_C as in Figure III.3, two instances in $rep(T_C)$ are given in Figure III.4.

SUPPLIER	ADDRESS	PRODUCT	CON | y ≠ Romorantin
x	Romorantin	nails	x ≠ mimi
toto	y	bolts	true
zaza	y	nuts	true

Conditional table T_C

Figure III.3

SUPPLIER	ADDRESS	PRODUCT
lulu	Romorantin	nails
toto	London	bolts
zaza	London	nuts

SUPPLIER	ADDRESS	PRODUCT
toto	Paris	bolts
zaza	Paris	nuts

Two instances in $rep(T_C)$

Figure III.4

III.2 Tables and relational algebra

We will study representation systems based on these three types of tables : Codd tables, naive tables and conditional tables. Representation systems based on these tables were extensively studied in [IL1] with Ω being a subset of relational algebra. The following results were demonstrated there.

Theorem [IL1] : $\langle S, rep, \Omega \rangle$ is a strong representation system for S being the set of all conditional tables, and Ω consisting of projection, selection, union, join, difference and renaming.

$\langle S, rep, \Omega \rangle$ is a weak representation system for S being the set of all naive tables, and Ω consisting of projection, positive selection, union, join and renaming.

$\langle S, rep, \Omega \rangle$ is a weak representation system for S being the set of all Codd tables, and Ω consisting of projection and selection.

The conditional tables in [IL1] do not contain a global condition. The notion of conditional table with global condition (as found here) was introduced in [G] in order to handle dependencies. The result of [IL1] on C-tables was also extended in [G] to C-tables with global conditions.

The above result for naive tables is particularly attractive since the algorithms to perform queries on naive tables are essentially

the same as those for the complete case. The variables can be treated as domain values that are pairwise different and different from all "real" domain values. The attractiveness of the naive tables is though severely diminished, as we shall see later, by their weak ability to support update operations.

The following theorem shows the limitations of the tables for handling queries. (We shall present in the next section, limitations for handling updates.)

Theorem : <S,rep,Ω> is not a weak representation system if

a) S is the set of all naive tables and Ω contains

i) projection and selection, or
ii) projection and difference.

b) S is the set of all Codd tables and Ω contains

i) projection, selection and union, or
ii) projection and join, or
iii) selection and difference.

Parts a)i), b)i) and b)ii) were proved in [IL1]. Note that Codd tables can support only unary operations. (Parts a)ii and b)i really require the assumption that the database scheme consists of at least two relation schemes).

III.3 Updating tables

We now turn our attention to update operations on tables. In the previous section, we introduced seven very general classes of updates. In this section, we use tables to specify some particular incomplete instances. We now specify some particular updates on these incomplete instances.

We will present several kinds of updates. These updates are all specific cases of the updates introduced in the previous section : insertion, deletion, integration, modification, subjection and augmentation. An update will be specified by a table, a condition, or a dependency. We do not claim that every conceivable update can thereby be expressed. However, we believe that this leads to a quite powerful update language which can be used as a yardstick for update capabilities.

In the following, S is a set of tables (e.g., the set of C-tables).

We first define insertion.

Insertion : Let T be a table in some S. Then the insertion of T is the operation on incomplete instances defined by $ins_T(X) = X \cup rep(T)$ for each incomplete instance X.

We will be of course interested in applying ins_T to instances represented by tables in S. We now present an example of insertion of a C-table "into" a C-table.

Example : Consider the C-table T of Figure III.3. Suppose that we want to insert the fact that there is a supplier named lili in Tokyo who supplies an unknown product. This insertion is accomplished by taking the one-tuple C-table <lili, Tokyo, v, true> (the global condition equals true), and inserting it into $rep(T_C)$. The result of this insertion is the incomplete instance $rep(T'_C)$ where T'_C is the C-table in Figure III.5.

SUPPLIER	ADDRESS	PRODUCT	CON | y ≠ Romorantin
x	Romorantin	nails	x ≠ mimi
toto	y	bolts	true
zaza	y	nuts	true
lili	Tokio	v	true

Figure III.5

The next operation to be considered is deletion. In complete databases, deletion is defined by a condition. (The tuples to be deleted are all the tuples satisfying the condition.) In incomplete databases, we allow incompletely specified updates. Hence we need the notion of extended conditions.

An extended condition F is a couple <F1,F2> where F1 is a conjunction of extended elementary conditions of the form A=x, A=a, A≠x, or A≠a, and F2 is a global condition.

An example of extended condition is: <NAME=v,v=toto or v=zaza>.

An extended condition F= <F1,F2> defines a set of complete instances K_F in the following way : K_F = {{t | satisfies F'} | F' a valuation of F1 respecting F2} (A valuation F' of F1 respects F2 if F2 is true for the values assigned to variables in transforming F1 to F'.)

For example, the condition F above defines the set K_F = {{t | t[NAME] = toto},{t | t[NAME]=zaza}}.

We are now ready to define the deletion operation.

Deletion : Let F be an extended condition. Then del(F) is the operation on incomplete instances defined by : $del(F)(X) = X (-) K_F$, for each incomplete instance X.

We now illustrate the use of deletion. Suppose that we want to delete either toto or zaza from the database specified by the table T_C in Figure III.3. This can be accomplished by taking the extended condition F, as defined above, and performing the operation del(F)(rep(T_C)). A table representing the result of this operation is given in Figure III.6. The reader can easily verify that in any complete instance represented by the table below either the second or third tuple is discarded.

SUPPLIER	ADDRESS	PRODUCT	CON	y≠Romorantin and (v=toto or v=zaza)
x	Romorantin	nails	x≠mimi and x≠v	
toto	y	bolts	toto ≠ v	
zaza	y	nuts	zaza ≠ v	

Figure III.6

Modification on incomplete databases is defined in a manner similar to that of [AV1]. However, since we are dealing with incomplete information, we allow extended conditions (as for deletions) to be used.

Modification : Let F and F' be extended conditions. Then mod(F;F') is the operation on incomplete instances defined by

$$mod(F;F')(X) = H(X)$$

where H is the set of complete modifications obtained by valuating F and F' in mod(F;F').

We now illustrate the modification by an example. For the sake of simplicity, we use only complete conditions in the example.

Example : We want to make a modification to the database specified by the table T_C in Figure III.3 : Change the parts supplied by suppliers in Helsinki to screws. The corresponding modification is mod<ADDRESS=Helsinki; PART=screws>. The result of this operation on T_C is represented by the table in Figure III.7. Note that the table contains some disjunctive information. (If the address of toto is Helsinki, then the part supplied is screws, otherwise it remains unchanged.)

SUPPLIER	ADDRESS	PRODUCT	CON	y≠Romorantin
x	Romorantin	nails		x ≠ mimi
toto	y	bolts		y ≠ Helsinki
toto	y	screws		y = Helsinki
zaza	y	nuts		y ≠ Helsinki
zaza	y	screws		y ≠ Helsinki

Modification

Figure III.7

We now present integration.

Integration : Let T be a table in S. Then the integration of T is the operation on incomplete instances defined by

$$int_T(X) = X \cap rep(T)$$ for each incomplete instance X.

Next we consider subjection updates. The subjection that is considered here is a special case of the general subjection defined in the previous section, and is based on an extension of "equality generating dependencies".

In the following, we call dependency an equality generating dependency [BV] where constants may appear. The following sentence g is an example of a dependency :

g= ∀ t (t(SUPPLIER)= toto => t(ADDRESS)=Paris).

The knowledge expressed by this dependency is that the hitherto unknown address of toto is Paris. In the update, we want to maintain the knowledge that zaza's address is the same as toto's. This will be accomplished by the subjection update.

For a set G of dependencies, sat(G) is the set of complete instances satisfying G. Now we have:

Subjection : Let G be a set of dependencies. Then the subjection by G is the operation on incomplete instances defined by

$$subj_G(X) = X \cap sat(G)$$ for each incomplete instance X.

The following example illustrates this notion of subjection.

Example : Consider again the instance rep(T_C) for T_C as in Figure III.3. Consider also the dependency g defined above. Then $subj_g(rep(T_C))$= rep(T'_C) for T'_C given in Figure III.8. Note that the information x ≠ toto was also inferred. This is due to the fact that the address of the supplier x is Romorantin; hence x cannot be toto.

SUPPLIER	ADDRESS	PART	CON	x≠toto
x	Romorantin	nails		x ≠ mimi
toto	Paris	bolts		true
zaza	Paris	nuts		true

Subjection

Figure III.8

The structure of the information in dependencies is such that it will be more

convenient to introduce negative information through negative dependencies instead of through negative subjection. By a negative dependency we mean a formula obtained by replacing, in a dependency, all equalities at the right hand side of the implication by inequalities. The following sentence g' is an example of negative dependency :

g' = ∀ t (t(SUPPLIER)=toto => t(ADDRESS)≠Paris).

The definition of the subjection remains the same, but we shall call it a subjection with a negative dependency to distinguish the type of information introduced. For the next example we want to incorporate the knowledge expressed by the negative dependency g' above into the database specified by the table T_C of Figure III.3. The result is rep(T'_C), for T'_C in Figure III.9.

SUPPLIER	ADDRESS	PRODUCT	CON	y≠Romorantin and y≠Paris
x	Romorantin	nails	x ≠ mimi	
toto	y	bolts	true	
zaza	y	nuts	true	

Subjection by a negative dependency

Figure III.9

The last operation to be defined is augmentation. We augment the database specified by one table with the information specified by another table. The formal definition is as follows.

Augmentation : Let T be a table in S. Then the augmentation of T is the operation on incomplete instances defined by $aug_T(X) = X \cup rep(T)$ for each incomplete instance X.

III.4 Results

We are now ready to present the main results of the paper. These results characterize the ability of the three particular tables to support update and query operations. For a table-based update in a representation system <S, rep, Ω>, we assume that the table used to specify the update is also in S, and that there are no variables in common in the two tables involved. Also, the variables used in specifying an update have to be "fresh", i.e. not used before. In particular, one cannot use the same variable in two different updates in an expression.

Theorem : <S,rep,Ω> is a strong representation system for S being the set of all conditional tables, and Ω consisting of all relational algebra operators and all six update operations.

The theorem shows the strength of conditional tables. The various updates are treated in the proof of the theorem in the following way :

- Insertion, deletion and integration can be embedded in a slight extension of the relational algebra on C-tables.
- A simple algorithm to compute the effect of a modification on a table can be exhibited.
- The ability to support subjection follows from [G]. The result of a subjection can be computed as $Chase_G(T)$, where Chase is a generalization of the well known Chase algorithm [MMS].
- Subjection by a negative dependency can be obtained by a modification of this generalized chase.
- Augmentation is achieved by adding a condition x=0 to every tuple in T, and the condition x≠0 to all the tuples in T', and taking the (set theoretic) union of the two tables. Here x is a variable not appearing elsewhere in the tables. If we denote the global conditions for T and T' by G and G', the global condition for the new table will be (G and X=0) or (G' and X≠0).

If conditional tables are very powerful with respect to update operations, the same cannot be said about naive tables. Even if naive tables can support quite a large subset of relational algebra, they are not very well suited for update operations.

Theorem : <S,rep,Ω> is a weak representation system if S is the set of all naive tables, and Ω consists of projection, positive selection, union, join, renaming, insertion, integration and subjection by positive dependencies without constants.

Insertion and integration are obtained using relational union and intersection. A result in [IL2], which states that naive tables can be "chased" in the same way as tableaux [MMS], can be used for subjection. For the limits of naive tables, we have established the following negative result.

Theorem : <S,rep,Ω> is not a weak representation system if S is the set of all naive tables, and Ω contains

a) positive selection, projection and deletion (even with complete condition), or

b) positive selection, projection and modification (even with complete condition), or

c) positive selection, projection and augmentation or

d) positive selection, projection and (general) subjection.

The main reason for this negative result is that naive tables cannot handle disjunctive information. Negative subjections are not supported since we cannot express inequalities on the null-values.

Our next result indicates a (limited) power of Codd tables.

Theorem : <S,rep,Ω> is a weak representation system if S is the set of all Codd tables, and Ω consists of projection, selection, insertion, integration and deletion with complete condition.

Insertion and integration are achieved through union and intersection. Since the inserted or integrated table is independent from the operand, these operations are in effect unary, and thus weaker than their relational counterparts. Deletion is obtained by deleting all tuples that possibly match the deletion condition. "Possibly match" means that there exists at least one substitution of the null-values which makes the comparison true. For this purpose, the evaluation rules are given by: "@=a" = true and "@=@" = true.

Finally, we present some limitions of Codd tables.

Theorem : <S,rep,Ω> is not a weak representation system if S is the set of all Codd tables, and Ω contains

a) selection and modification (even with complete conditions), or

b) selection and subjection, or

c) selection and augmentation.

This result is a consequense of the facts that Codd tables cannot support disjunctive information. The failure to support subjections follows from the fact that we can neither equate two unknown values (which we could do with naive tables), nor express inequalities between them.

IV CONCLUSIONS

The results obtained in the last section were not surprising: more complicated tables can support more update operations. However, the results are quite discouraging for naive tables : Naive tables can handle queries in a simple and elegant manner, but they are quite inadequate with respect to updates. Indeed, only C-tables could support updates in a reasonable fashion. This strongly suggests that C-tables should be used as the basis for an implementation of incomplete databases.

C-tables can be criticized for the complexity of query computation. Two solutions can be adopted :

(1) The database is stored as a C-table, but for query purposes, only the naive part of the table is used. (Answer to queries are therefore only approximations.) All the information of the C-table is used for updates. Hence, the content of the database is always viewed as close as possible to the known reality.

(2) If conditions are assumed to be rare, then the complexity argument against C-tables does not hold anymore. A physical implementation can then be used offering the full advantage of C-tables, but treating most of the information with tools developped for naive tables.

The first solution may be chosen for applications where queries are much more frequent than updates if an approximation of the correct answer is acceptable. The second solution should be prefered if conditions are essentially used to handle exceptions, and are thus quite rare.

A final remark on the semantic nature of update operations should be made. The semantics we have used is in essence algebraic in that we consider updates as operations, i.e. as mappings from databases to databases. Another approach which appeared in the literature is the proof theoretic one [FUV,KUV]. In the proof theoretic approach, the update is specified by some properties that the resulting database should satisfy. The method is thus completely unconstructive. (One can note that databases specified by tables can be viewed as a set of axioms of a first order theory (see [I]). However, since we deal with closed world databases, updates are a form of non-monotonic reasoning, and thus the methodology proposed for instance by [FUV] is not applicable). The difference between the "algebraic" and the "proof theoretic" approaches is somewhat similar to the difference between relational algebra and relational calculus. An interesting topic for further research is the comparison of the two approaches in search of some sort of equivalence between them.

REFERENCES

[AH] Abiteboul S., R. Hull, Update Propagation is Semantic Database Models, (1984), Proceedings of the International Conference on Foundations of Data Organization, Kyoto (1985).

[AV1] Abiteboul, S., V. Vianu, Transactions in Relational Databases. Proc. 10th Internat. Conf. on Very Large Data Bases, Singapore (1984).

[AV2] Abiteboul, S., V. Vianu, Transactions and Constraints. Proc. ACM SIGACT - SIGMOD Symp. on Principles of Database Systems, Portland, Oregon, (1985).

[BS] Bancilhon, F., N. Spyratos, Update Semantics of Relational Views. ACM Trans. Database Syst. 6,4 (1981).

[BV] Beeri, C., M. Vardi, Formal Systems for Tuple and Equality Generating Dependencies. SIAM. J. Comput. 13, 1 (1984).

[B] Biskup, J., A Foundation of Codd's Relational Maybe - operations. ACM Trans. Database Syst. 8, 4 (1983).

[C] Codd, E.F., Extending the Relational Model to Capture more Meaning. ACM Trans. Database Syst. 4, 4 (1979).

[FUV] Fagin, R., J.D. Ullman, M. Vardi, On the Semantics of Updates in Databases. Proc ACM SIGACT-SIGMOD Sympφ. on Principles of Database Systems, Atlanta (1983).

[G] Grahne, G., Dependency Satisfaction in Databases with Incomplete Information. Proc. 10th Internat. Conf. on Very Large Data Bases, Singapore (1984).

[I] Imielinski, T., On Algebraic query processing in logical databases. In Advances in Database Theory, vol. 2 (H. Gallaire, J. Minker and J.M. Nicolas, eds.) (1984).

[IL1] Imielinski, T., W. Lipski, Incomplete Information in Relational Databases. J. Assoc. Comput. Mach. 31, 4 (1984).

[IL2] Imielinski, T., W. Lipski, Incomplete Information and Dependencies in Relational Databases. Proc. ACM SIGMOD Internat. Conf. on Management of Data, San Jose (1983).

[KUV] Kuper, G., J.D. Ullman, M. Vardi, On the Equivalence of Logical Databases. Proc. ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, Waterloo, Ontario, (1984)

[MMS] Maier, D., A.O. Mendelzon, Y. Sagiv, Testing Implication of Data Dependencies. ACM Trans. Database Syst. 4, 4 (1979)

[R1] Reiter, R, On Closed World Databases. In 'Logic and Data Bases' (H. Gallaire and J. Minker, eds.), Plenum Press, New York (1978)

[R2] Reiter, R., A Sound and Sometimes Complete Query Evaluation Algorithm for Relational Databases with Null Values. Techn. Rep., Dept. of Computer Science, Univ. of British Columbia, Vancouver, BC (1983)

[U] Ullman, J. D., Principles of Database Systems, Second Edition. Computer Science Press, Potomac, MD (1982)

[Vr] Vardi, M., Querying Logical Databases, Proc. ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, Portland, Oregon, (1985).

[Vs] Vassiliou, Y., Null Values in Data Base Management, a Denotational Semantics Approach. Proc. ACM SIGMOD Internat. Conf. on Management of Data, Boston, MA (1979).