INCOMPLETE INFORMATION AND DEPENDENCIES IN RELATIONAL DATABASES*

Tomasz IMIELINSKI
McGill University
and
Witold LIPSKI, Jr.
Polish Academy of Science

Abstract

An incomplete information relational database combines two types of information about the real world modeled by the database: (a) the information represented by tables with null values ("value not known") allowed as entries, and (b) the data dependencies, which are known to be satisfied in the real world. We view the well known chase procedure as a process which transforms type (b) information into an "equivalent" type (a) form. Assuming that the data dependencies are arbitrary implicational dependencies, we show that this transformation is not quite equivalent, but the corruption of information introduced cannot be discovered if the query language uses the operations of projection, positive selection (i.e. no negation in selection condition), union, natural join and renaming of attributes. This result can be interpreted also as the new important property of chase.

The influence of so-called view dependencies on the table with null values is also examined.

Introduction

An incomplete information relational database combines two types of information about the real world modeled by the database:

(a) the information represented by a collection of tables with null values ("value not known") allowed as entries, and

(b) the data dependencies, which are known to be satisfied in the real world.

We assume that null values are variables from a set V, so that it is possible to represent - by using two occurrences of the same variable - the fact that two values, though unknown, are known to be equal (see [IL1, IL2]). The data dependencies are assumed in this paper to be arbitrary implicational dependencies [BV] ("generalized dependencies" in [U11]), i.e. equality generating dependencies which generalize functional dependencies, and total tuple generating dependencies which generalize functional dependencies, and total tuple generating dependencies which generalize multivalued and join dependencies.

Both types of information can be used in the process of responding a query. In this paper we view the well known chase procedure [MMS, BV] applied to an incomplete information relational database as a process which transforms type (b) information into an "equivalent" type (a) form.

We show that this transformation is not quite equivalent, but the corruption of information introduced cannot be discovered if the query language does not use operations other than projection, positive selection (i.e. no negation in selection condition), union, (natural) join and renaming of attributes (this includes arbitrary conjunctive queries).

The problems of interference between incomplete information and database dependencies were treated in several papers (e.g. [V]). In those papers the problem of "satisfaction" of database dependency by the table with null values was investigated. In our belief such a statement of the problem is incorrect, this is not the table but the unknown state of reality which is to satisfy dependencies. In fact database dependencies "improve" our knowledge about the unknown state of reality restricting the set of possibilities.

Formally, suppose then that a state of an incomplete information database consists of a single table T (built up from constants and variables) and a set Σ of dependencies. Denoting by Rep(T) the set of possible states of the real world represented by T, and by Sat(Σ) the set of all relations satisfying all dependencies in Σ, our knowledge about the real world - i.e. the set of all possibilities which cannot be ruled out on the basis of our knowledge - is given by the intersection Rep(T) $\cap$ Sat(Σ).

The problem is whether this knowledge could be represented or approximated by some other table itself. In Section 2 the main result of the paper shows in what extent it is possible. In Section 3

---

* Preliminary Version

we describe also how the knowledge that the table with null values is the state of some view of the database (so-called view dependency [KP]) could be incorporated into this table content. Finally we briefly discuss possible influence of the other types of dependencies (like inclusion dependencies).

1. Basic Notions

1.1 Relations and Tables

A relation r is a collection of functions (tuples) from a set of attributes (A) to some domain (D). For notational simplicity we assume one common (infinite) domain. By the type of a relation r we mean its set of attributes $\alpha(r)$. By a relation name we mean a symbol R with an associated type. We will say that relation r is the instance of relation name R is the type of r is equal to the type of R. We consider the usual relational operators: projection, join, union, and selection (denoted by $\pi_Y(r)$, $r \bowtie s$, $r \cup s$, and $\sigma_E(r)$, respectively). If E does not contain negation, then $\sigma_E(r)$ will be called positive selection. For any subset of relational operators project (P), join (J), selection (S), positive selection (S^+), and union (U), by a relational Ω-expression we mean any well formed expression built up from relation names and the relational operators in Ω. We will talk, for example, about PJ-expressions built up from projection and join. By the type of a relational expression f we mean the set of attributes of the result of this expression and we denote it by $\alpha(f)$.

A database scheme P is the collection of relation names $R_1, \ldots, R_n$ together with a set of dependencies. The instance of the database scheme is the sequence of relations $\langle r_1 \ldots r_n \rangle$ and it is called multirelation. The type $\alpha(r)$ of multirelation is defined as $\langle \alpha(r_1) \ldots \alpha(r_n) \rangle$.

A table [IL1] is a relation with variables as well as constants allowed as entries. Let V be a countably infinite set of symbols called variables. By a table of type X we mean any finite collection of tuples (i.e. functions from X into D ∪ V). By a valuation, we mean any mapping v:V->D. A valuation can be extended over the set D ($v(c)=c$ for all c in D) and over the set of tuples and tables in the standard way. By Rep(T) we will mean the function assigning to each table T the following set of relations:
{ s : there exists v such that $v(T) \subseteq s$}

By a multitable we shall mean a sequence of tables $T = \langle T_1 \ldots T_n \rangle$. Tables and multitables are just relations and multirelations with null values. The variables play the role of indexed null values. Therefore it is possible to represent the fact that the same two values although unknown are the same. The type of multitable is defined analogously as the type of multirelation.

Example 1

The multitable $T = \langle T_1, T_2 \rangle$ where

T_1 =

Supplier	Address
S_1	x
S_2	y

T_2 =

Supplier	Product
S_1	z
S_2	z

represents the fact that two suppliers with unknown addresses supply the same (although unknown) product.

Relational Expressions and Tables

In [IL1] the criterion for semantically meaningful extensions of relational expressions over tables was introduced. It was stated that the set of relational Ω-expressions is correctly extendable over the set of tables if for any table T and any relational Ω-expression f

$$\mathrm{Rep}(f(T)) \underset{\Omega}{\equiv} f(\mathrm{Rep}(T))$$

where f(T) denotes the resultant of f, $f(\mathrm{Rep}(T)) = \{f(s) : s \in \mathrm{Rep}(T)\}$ and $\underset{\Omega}{\equiv}$ stands for Ω-equivalence of two sets of relations, $\mathfrak{X} \underset{\Omega}{\equiv} \mathfrak{Y}$ iff for any Ω-expression f, $\cap f(\mathfrak{X}) = \cap f(\mathfrak{Y})$. If it is clear from the context what is the set of relational expressions which we have in mind, we will skip Ω under the $\equiv$ equivalence symbol. In other words the above condition says that Rep(f(T)) and f(Rep(T)) are all indistinguishable by Ω-expressions. In [IL1] we showed that relational PS^+UJ-expressions can be correctly extended over tables i.e. that

$$f(\mathrm{Rep}(T)) \underset{PS^+UJ}{\equiv} \mathrm{Rep}(f(T))$$

simply by treating variables as pairwise distinct constants, distinct also from all domain elements.

Example 2

Let T =

A	B	C
x	y	c
a	y	v
a'	z	c'

The relational algebra expression

$$f(R) = \sigma_{(A=a) \vee (A=a')} (\pi_{AB}(R) \bowtie \pi_{BC}(R))$$

evaluated over the table T, yields the calculation:

$\pi_{AB}(T) \bowtie \pi_{BC}(T)$ =

A	B	C
x	y	c
x	y	v
a	y	c
a	y	v
a'	z	c'

and f(T) =

A	B	C
a	y	c
a	y	v
a'	z	c'

The $\equiv$ equivalence plays a crucial role in the process of incorporating implicational dependencies into the table content.

1.2 Dependencies

We will deal here with the most general, implicational dependencies generalizing functional and join dependencies. Implicational dependencies can

be classified into equality generating and total tuple generating dependencies (see [BV]. In fact our definitions are even slightly more general.

By a total tuple generating dependency (ttgd) we mean any statement of the form Q/t where $Q = \langle Q_1 \ldots Q_n \rangle$, t are multitables, $t = \langle t_1 \ldots t_n \rangle$, such that for each i, t_i consists of at most a single tuple and all variables occuring in t occur in Q. An equality generating dependency (egd) is any statement of the form $Q/e_1=e_2$ where e_1, e_2 are two symbols at least one of them a variable, occuring in Q.

The multirelation $r = \langle r_1 \ldots r_n \rangle$ satisfies ttgd Q/t iff for any valuation v if $v(Q_i) \subset r_i$ then $v(t_i) \subset r_i$ ($v(t_i)$ is a relation consisting of only single tuples) for i = 1 ... n.

In the same way, the multirelation $r = \langle r_1 \ldots r_n \rangle$ satisfies egd $Q/e_1=e_2$ iff for any valuation r if $v(Q_i) \subset r_i$ for i = 1 ... n then $v(e_1) = v(e_2)$.

It is easy to see that ttgd generalizes multivalued and join dependencies, while egd generalizes functional dependencies.

Let Σ be the set of implicational dependencies, by Sat(Σ) we will denote the set of all multirelations satisfying the dependencies from Σ.

Example 3

The ttgd Q/t where

$$Q = \langle Q_1, Q_2 \rangle, \quad t = \langle t_1, t_2 \rangle$$

where

Q_1 =

A	B	C
a	b	x
y	b	c

Q_2 =

B	C	D
b	w	d
z	c	d

and

t_1 =

A	B	C
a	b	c

t_2 =

B	C	D
b	c	d

correspond to two multivalued dependencies B↦↦A and D↦↦C satisfied in the schema $P = \langle R_1, R_2 \rangle$ where $\alpha(R_1) = ABC$, $\alpha(R_2) = BCD$ and B↦↦A is satisfied by every instance of R_1 while D↦↦C by every instance of R_2.

In the same schema functional dependency B→C to be valid in R_1 could be represented by the following egd $Q/e_1=e_2$) $Q = \langle Q_1, Q_2 \rangle$

Q_1 =

A	B	C
x	z	v
y	z	w

$Q_2 = \emptyset$

$e_1 = v$ and $e_2 = w$.

Chase

The chase procedure for tables is defined analogously to the chase for tableaux. Let Σ be the set of implicational dependencies. Basically speaking the chase procedure is the sequence of application of two types of rules generated by ttgd or egd from Σ to the multitable T. (We assume that the type of the multitable Q for any implicational dependency in Σ is equal to the type of T).

Each rule application consists of two phases. First the pattern corresponding to the multitable Q, in T is identified, that is, some substitution λ such that $\lambda(Q) \subset T$. Then depending on whether the given dependency is ttgd or egd, the new "multituple" $\lambda(t)$ is added or two symbols $\lambda(e_1)$ $\lambda(e_2)$ are equated. The procedure terminates when we cannot apply further any rule for any of the dependencies from Σ, i.e. we reach some "saturated" fixpoint due to the above rules.

The result of the chase procedure for the multitable T and the set Σ of implicational dependencies will be denoted by $chase_\Sigma(T)$.

2. Main Result

We are now going to present our main result concerning the possibility of incorporating implicational dependencies into the table content. This result could be also interpreted as the new property of already known chase procedure.

Theorem 1

Let Σ be the set of implicational dependencies and let T be a multitable then

(i) $Rep(T) \cap Sat(\Sigma) \equiv Rep(chase_\Sigma(T))$, where the equivalence is understood as PS$^+$UJ equivalence or even the equivalence with respect to the set of relational expressions generated by projection, positive selection, restriction, union, join and renaming of attributes.

Before sketching the proof let us formulate some remarks.
The intuitive conclusion from the theorem is that we may forget about dependencies while computing responses to queries provided the database is always maintained in a "chased" form.

The fact that in general the equivalence (i) cannot be replaced by equality can be illustrated by the following two examples.

Example 4. Let

T =

NAME	ADDRESS
x	London
y	Paris

(x, y are variables), and let Σ consist of a single functional dependency NAME → ADDRESS. Then $chase_\Sigma(T) = T$ and

R =

NAME	ADDRESS
Smith	London
Smith	Paris

is in $Rep(chase_\Sigma(T))$ but not in $Rep(T) \cap Sat(\Sigma)$.

Example 5. Let

T =

COMPANY	LOCATION	PRODUCT
McDonald	New York	Hamburger
x	Boston	Cheeseburger

and let Σ consist of a single multivalued dependency COMPANY ↦↦ PRODUCT. Then $chase_\Sigma(T) = T$ and

	COMPANY	LOCATION	PRODUCT
R =	McDonald	New York	Hamburger
	McDonald	Boston	Cheeseburger

is in $Rep(chase_\Sigma(T))$ but not in $Rep(T) \cap Sat(\Sigma)$.

Intuitively, the fact that we do not have equality in (1) can be explained as follows. Any implicational dependency, when applied in a "forward manner", implies either the existence of a tuple or the equality of symbols, which is then represented in the result of the chase. We may however apply an implicational dependency in a "backward manner", which produces (a disjunction of) inequalities between symbols, which cannot be represented in the chase process.

Sketch to proof of the theorem

Let Σ_t denote the subset of all ttgds from Σ. Let γ denote the "final" substitution of symbols from T in $chase_\Sigma(T)$, i.e. $\gamma(x) = chase_\Sigma(x,T)$; the symbol into which x is transformed during the process of chasing T by Σ. Notice that the substitution γ is the direct result of application of rules corresponding to egd's (although some substitutions may emerge in consequence of applying some ttgd's rules before). By $\gamma(T)$ denote the table resulted from T by substituting for any symbols x occurring in T $\gamma(x)$.

The proof of the theorem proceeds in the following stages

(1) $Rep(chase_\Sigma(T)) = Rep(chase_{\Sigma_t}(\gamma(T)))$

(2) $Rep(chase_{\Sigma_t}(\gamma(T))) \equiv Rep(\gamma(T)) \cap Sat(\Sigma)$

(3) $Rep(\gamma(T)) \cap Sat(\Sigma) = Rep(T) \cap Sat(\Sigma)$

The formula (1) follows from $chase_\Sigma(T) = chase_{\Sigma_t}(\gamma(T))$ a simple property of the chase process.

The crucial part is (2). It follows from three facts. First, for any relation R and any set of ttgd's Σ_o, there exists the smallest R* containing R and satisfying ttgd's from Σ_o. Moreover, this R* could be computed as $\bigcup_{i=0}^{\infty} g^i(R)$, the union being in fact finite, for some relational expression g involving projection, positive selection, restriction, union, join or renaming of attributes.

The second fact is already mentioned (see 1.1) property of extension of relational algebra to tables with null values i.e. that $Rep(g(T))) \equiv g(Rep(T))$ for any g described above and "$\equiv$" understood as Ω_o-equivalence for Ω_o consisting of projection, union, positive selection, join, and renaming of attributes.

The third fact is that

(2.1) $Rep(T) \cap Sat(\Sigma_o) \underset{\Omega_m}{\equiv} \{\bigcup_{i=0}^{\infty} g^i(R) : R \in Rep(T)\}$

for Σ_o - the set of ttgd's, Ω_m the set of all monotonic (positive) relational operations (i.e. all operations in spite of difference). In order to prove it, it suffices to show that both sets have the same minimal elements. Indeed, in the case of monotonic Ω_m-expressions $\cap f(\mathbb{R}) = \cap f(\mathbb{R}_m)$ where $\mathbb{R}_m$ denotes the subset of minimal elements of $\mathbb{R}$ ($S_1 \subset S_2 \quad f(S_1) \subset f(S_2)$ for any Ω_m-expressions).

First of all the righthand side set is contained in the lefthand side one, hence $\bigcup_{i=0}^{\infty} g^i(R) \in Sat(\Sigma_o)$ for any R, therefore it is sufficient to show that any minimal element on the righthand side set belongs to the righthand one. Let $S \in Rep(T) \cap Sat(\Sigma_o)$ then $S \supseteq v(T)$ for some valuation v, and certainly $S \supseteq \bigcup_{i=0}^{\infty} g^i(v(T))$. If S is the minimal element the equality takes place and certainly $\bigcup_{i=0}^{\infty} g^i(v(T))$ belongs by the definition to the righthand side set.

Now we may show how to prove (2). Let us observe that $chase_{\Sigma_o}(T)$ for Σ_o bring a set of ttgd's can be represented also as $\bigcup_{i=0}^{\infty} g^i(T)$ for the g being relational expression defined above. Indeed, the process of chase-computation in the case of ttgd's is just the process of computing the smallest T* containing T and satisfying Σ, where variables are treated as additional domain elements (different from all proper domain elements). Therefore, we may apply fact 1 here.

Moreover, in light of the second fact, we have

(2.2) $Rep(\bigcup_{i=0}^{\infty} g^i(T)) \equiv \bigcup_{i=0}^{\infty} g^i(Rep(T)) =$

$\{\bigcup_{i=0}^{\infty} g^i(R) : R \in Rep(T))\}$,

hence $\bigcup_{i=0}^{\infty} g^i(\mathbb{R}) = \bigcup_{i=0}^{k} g^i(\mathbb{R})$, for some k, is the relational expression of the same type (Ω), than g (union is allowed).
From the above equivalences (2.1) and (2.2) we obtain

(2.3) $Rep(chase_\Sigma(T)) \underset{\Omega_o}{\equiv} Rep(T) \cap Sat(\Sigma)$

(from (2.1) and (2.2) we infer

$Rep(chase_\Sigma(T) \underset{\Omega_p}{\equiv} Rep(T) \cap Sat(\Sigma)$, and since

$\Omega_o \subset \Omega_p$, (the set of all positive operations); we obtain (2.3) as the special case). From (2.3) we infer $Rep(chase_{\Sigma_t}(\gamma(T)) \equiv Rep(\gamma(T)) \cap Sat(\Sigma_t)$

where Σ_t is the subset of all ttgd's in the original Σ. Finally we obtain (2) from the fact that $Rep(T) \cap Sat(\Sigma) = Rep(\gamma(T)) \cap Sat(\Sigma)$, and that $Rep(\gamma(T)) \cap Sat(\Sigma) \underset{\Omega_o}{\equiv} Rep(\gamma(T) \cap Sat(\Sigma_t)$.

The last equivalence can be proved using the notion of so-called conditional table (see [I] or [IL1]). Part (3) of the proof is straightforward. Substitution of γ eliminates only those valuations of T which violate equality generating dependencies in Σ, therefore we do not loose any relations in $Rep(T) \cap Sat(\Sigma)$, hence (3) is true.

Comment

In the above sketch to proof of theorem 1 we presented the full proof for the "ttgd's part" of

Σ because it does not refer to the notion of conditional table. This notion is needed when proving the part of the theorems in which egd's play important roles (i.e. that

$Rep(\gamma(T)) \cap Sat(\Sigma_t) \equiv Rep(T) \cap Sat(\Sigma))$.

We did not present this proof because it needed some basic notions not introduced in this paper. This result is taken from [I].

3. View Dependencies and Incomplete Information

The notion of the view dependency was introduced in [KP]. Basically speaking view dependency is defined as the constraint on the possible states of the view f resulted from the way f was computed. If the basic schema satisfied some (implicational) dependencies Σ, then they could interact with relational expression f, resulting in some combined constraint f_Σ on the possible states of view f.

By $f_\emptyset$ we will denote the "pure" view dependency constraint on the state of the view f computed over "free" basic schema ($\Sigma = \emptyset$). Obviously

$Sat(f_\Sigma) = \{f(R) : R \in Sat(\Sigma)\}$ and

$Sat(f_\emptyset) = \{f(R) : R \in \mathfrak{R}\}$ where $\mathfrak{R}$

is the set of all possible instances of the free basic schema. The interesting example of view dependency is just an universal instance assumption (UIA) where f is the sequence of projections.

Suppose now that the same table T with null values (variables) represents the state of the view f of the basic schema satisfying some dependencies. from Σ. Is it possible to incorporate the additional knowledge given in the form of view dependency f_Σ into the table content in the analogous way we did in the previous section? The importance of the problem can be explained as similarily as before, if it is possible we could evaluate queries (to the view) basing only on the table content and "forgetting" about view dependencies.

Example 6

Let $f(R) = \pi_{ACD}(\pi_{AB}(R) \bowtie \pi_{BCDE}(R))$

and let

$\Sigma = \{(A \to B), (B \to C);$ $\begin{matrix} abx_1x_2x_3 \\ a_1x_4cx_5x_6 \\ x_7x_8cde \end{matrix}$ / $abcde\}$

(Instead of using egd's notation here, for simplicity we use "classical" f.dependency notation, ttgd corresponds to the lossless join AB $\bowtie$ AC $\bowtie$ CDE.)

Let T = | A | C | D |
|---|---|---|
| a | c | d |
| a | x | e |
| b | x | f |

be the table

with null value x - the state of the view of schema f. The question is whether from this fact we can infer some additional knowledge and incorporate it in the table content.

In this section we will show that it is possible to incorporate this kind of additional knowledge in the table content. We will use here the "inverted relational expression" technique described in [IL2].

The basic fact in this technique is that tables can serve as tools for representing inversions of relational expressions. This is the thesis of the following theorem.

Theorem 2

Let f be a relational PJ-expression and let T be a table, then there exists a table U such that $Rep(U) = f^{-1}(Rep(T)) = \{s : f(s) \in Rep(T)\}$. In particular, if T is a relation (table with no variables), then $Rep(U)) = f^{-1}(Rep(T)) = \{s : T \subseteq f(s)\}$. In general, there are many such tables U. We shall informally denote such a table U constructed by the algorithm given in [IL2] by $f^{-1}(T)$, so that $Rep(f^{-1}(T)) = f^{-1}(Rep(T))$. If $T = \{t\}$ we will frequently write $f^{-1}(t)$ instead of $f^{-1}(\{t\})$. Basically speaking the algorithm for computing $f^{-1}(T)$ (up to variable renaming) presented in [IL2] is a recursive procedure similar to that of tableau construction. (This is no surprise since tableau can be viewed as the representation of the inversion of the relational expression to which it corresponds.) The procedure has two basic steps:

(i) if $f = g \bowtie h$ then INVERSE :=INVERSE $(g, \pi_{\alpha(g)}(T)) \cup$ INVERSE$(h, \pi_{\alpha(h)}(T))$.

(ii) if $f = \pi_Y g$ then extend T by adding new columns corresponding to attributes in $\alpha(g) \setminus Y$ (set difference). Fill these columns by distinct new variables. Call the resulting table W
INVERSE :=INVERSE(g,W).

In fact we showed an even slightly stronger property, namely that

(2) $Rep(u(f^{-1}(T)) = f^{-1}(Rep(u(T))$
for every valuation u: of variables from V(T) (the set of variables occurring in T).

The fact that we are able to represent inversions of relational expressions by the tables is very useful in solving our present problem as the following theorem shows.

Theorem 3

Let f be the relational expression built up from projection, join and positive selection. Let the table T be the state of the view f of the basic database schema satisfying the set Σ of implicational dependencies. Then the following equivalence takes place

$Rep(f(chase_\Sigma f^{-1}(T))) \equiv Rep(T) \cap Sat(f_\Sigma)$

where f is computed on the table T, as before, treating variables as domain elements which are pairwise different and besides that different from any "proper" domain elements, and equivalence "$\equiv$" is understood as Ω-equivalence for $\Omega = PS^+J$.

Proof

By the theorem 2

$Rep(f^{-1}(T)) = f^{-1}(Rep(T))$ and

$Rep(chase_\Sigma(f^{-1}(T)) \equiv f^{-1}(Rep(T) \cap Sat(\Sigma)$

by theorem 1, now

$Rep(f(chase_\Sigma(f^{-1}(T)) \equiv f(Rep(chase_\Sigma(f^{-1}(T))$

by the property of extension of relational expressions over tables (fact two in the proof of theorem 1).

From these three equivalences we obtain

$Rep(f(chase_\Sigma(f^{-1}(T)))) \equiv f(f^{-1}(Rep(T) \cap Sat(\Sigma))) =$

$Rep(T) \cap Sat(f_\Sigma)$

Conclusion

The table $f(chase_\Sigma(f^{-1}(T)))$ is the table resulted from T by incorporating additional knowledge given in the form of constraint f_Σ into the table content. Now, again, as in the previous situation we may "forget" about the additional constraint when processing the queries.

Example 7

We will continue here with the previous example and compute the table $f\ chase_\Sigma(f^{-1}(T))$.

This is done (as the structure of this expression shows) in three phases.

(1) Compute $f^{-1}(T)$

We apply the algorithm given by theorem 2, treating variables in T as different, additional domain elements.

$f^{-1}(T) =$

A	B	C	D	E
a	y_1	w_1	w_2	w_3
w_4	y_1	c	d	w_5
a	y_2	w_1'	w_2'	w_3'
w_4'	y_2	x	e	w_5'
b	y_3	w_1''	w_2''	w_3''
w_4''	y_3	x	f	w_5''

(2) Compute $chase_\Sigma(f^{-1}(T))$

We apply functional dependencies (A→B), (B→C) obtaining the substitution of the table T.

A	B	C	D	E
a	y_1	c	w_2	w_3
w_4	y_1	c	d	w_5
a	y_1	c	w_2'	w_3'
w_4'	y_1	c	e	w_5'
b	y_3	c	w_2''	w_3''
w_4''	y_3	c	f	w_5''

Now we apply ttgd (corresponding to the lossless join AB ⋈ AC ⋈ CDE, we obtain (between the others) additional tuples

A	B	C	D	E
a	y_1	c	d	w_5
a	y_1	c	e	w_5'
b	y_3	c	d	w_5''
b	y_3	c	e	w_5'

(3) Compute $f(chase_\Sigma(f^{-1}(T))$

After some simplifications we obtain

$f\ chase_\Sigma(f^{-1}(T)) =$

A	C	D
a	c	d
a	c	e
b	c	f
b	c	e
b	c	d

The resulted table consists of considerably more precise information than the original one.

Remarks

As the second example we may take database schema satisfying UIA (universal instance assumption). This is also view dependency and can be treated in the same way first performing the inversion of the multiexpression $<\pi_{x_1}(R), \ldots, \pi_{x_n}(R)>$, then chase, and then the multiexpression $<\pi_{x_1}(R), \ldots, x_m(R)>$ again. This kind of similar construction has been proposed in frame of so-called weak universal instance assumption (see [V]).

Our technique here is of course more general and is not restricted to this very special case.

4. The Other Types of Dependencies

Until that moment incorporation of the dependencies into the table content resulted in decreasing (or at least not increasing) of the number of null values in the table. It is true for implicational dependencies but of course need not be true in general (in general, of course, any formula can form integrity constraint).

Consider so-called implicational dependencies, the incorporation of this kind of knowledge into the table content would rather introduce new types of null values than decrease its number.

Example 8

Let R =

A	B
a	b
a'	b

and S =

B	C
b	c

and let implicational dependency be of the form

$\pi_B(R) \subset \pi_B(S)$, then

certainly, we may incorporate this kind of additional knowledge into the current state (which is in fact multirelation, without null values) by creating additional tuple <b', x> and adding it to the table S. The result is

introduction of the new null value. It is an interesting problem to examine the global problem of incorporating both implicational and inclusion dependencies into the table content. This problem will be treated elsewhere in the full version of the paper.

5. Conclusions and Some General Remarks

The problem examined here could be also formulated more generally as the problem of proportion between so-called extension and intension of the database. The extension of the database is usually understood as the current state of the database while intention is rather referred to integrity constraints or time independent properties of the database.

In the ordinary relational database this distinction is sharp. While relations (extension) serve as the basis for query answering, intention (integrity constraints) are used only for checking the consistency. Such a nice separation does not hold any more when we deal with more complex "indefinite" databases, like those with incomplete information or containing "general rules" given in the form of Horn clauses [GMS], [K].

In such a database intension or time independent information (formulas) play double roles, the first which we will call passive is integrity constraint, or "guard" of consistency), the second which we will call active is the role in influence on the process of query evaluation.

These two roles have been distinguished in the literature as the role of integrity constraint and the role of general rule ("procedure") for computing data. It is the responsibility of the database designer to assign the roles to time independent formula.
Making the formula passive has the consequences on the database extension, intuitively we must represent "active part" of the formula in the extension in order to rule-out this formula from the query answering process.

On the other hand, leaving the "intension" formula active is more like leaving a hidden rule for computing data which must be consulted every time during the query answering process.

This is the disadvantage, but in the same time we save space not storing the redundant facts which could be simply derived.

In the general case of database applications, the first approach seems to be more practical however. The reason is simple, usually database extension is represented in some form (relation, table) which is easy to manipulate on in some uniform manner (e.g. relational algebra). The cost of this easy manipulation is paid by the inherent redundancy of the database (for example introduced by multivalued dependency).

This solution was adopted here, in context of tables with null values and large class of dependencies.

References

[BV] Beeri, C., Vardi, M.Y., Formal systems for tuple and equality generating dependencies. To appear in SIAM J. Comput.

[GMN] Gallaire, H., Minker, J., Nicolas, J.M., Logic and databases, unpublished manuscript, Maryland, 1982.

[I] Imielinski, T., Problems of representing information in the relational database, Doctoral Thesis, Warsaw, 1981.

[IL1] Imielinski, T., Lipski, W., On representing incomplete information in a relational database, Proc. 7th Internat. Conf. on Very Large Data Bases, Cannes, France, Sept. 1981, pp. 388-397.

[IL2] Imielinski, T., Lipski, W., Incomplete information in relational data bases, ICS PAS Report 475, May, 1982.

[IL3] Imielinski, T., Lipski, W., Inverting relational expressions, uniform and natural technique for various database problems Proc. of ACM PODS Conf., Atlanta, Mar. 1983.

[K] Kowalski, R., Logic for data description in logic and databases, eds. Gallaire, Minker, 1979.

[KP] Klug, A., Price, R., Determining view dependencies using tableaux. ACM TODS Vol. 7, No. 3, Sept. 1982, pp. 361-380.

[MMS] Maier, D., Mendelzon, A.O., Sagiv, Y., Testing implications of data dependencies, ACM Trans. on Database Syst. 2 (1977), 201-222.

[Ull] Ullman, J., Principles of Database Systems, second edition. Computer Science Press, Potomac, MD 1982.

[V] Vassiliou, Y., Functional dependencies and incomplete information, Proc 6 VLDB, Montreal, Canada, Oct. 1980, pp. 260-269.