A Unified Treatment of Null Values using Constraints *

Kasım S. Candan[†] **John Grant**[‡] **V.S. Subrahmanian**[§]

Abstract

An important reality when studying relational databases is the fact that entries in relational tables may often be "missing" or only partially specified. The study of such missing information has led to a rich body of work on "null values." It was recognized early on that there are many different types of null values, each of which reflects different intuitions about why a particular piece of information is missing. Different relations (or even the same relation) could contain different types of null values; yet, very little work has been done on providing a unifying model that reasons with different types of nulls. In this paper, we use constraints to provide a unifying framework for the most common types of nulls. We show how viewing tuples containing null values of these types can be viewed as constraints, and how this leads to an algebra for null values. In particular, this algebra contains a unique operator (called the "compaction" operator) used to remove redundancies from null valued relations. We have studied various properties of this algebra. We have built a prototype implementation based on the null valued operators described here and conducted various experiments using this testbed.

1 Introduction

The relational model of data mandates that all information be stored as relational tables where the rows are called *tuples* and the columns represent attributes. Often entries are missing, or are only partially specified in the tables for a variety of reasons. Such missing values are called *null values*, and over the last twenty years, a vast body of literature has been devoted to the study of different types of null values. Let us quickly consider a simple example in order to determine some of the different types of null values.

Example 1.1 Consider a simple relational database that contains three relations – a `phone` relation that has the schema (`Name`, `Phone`), a `spouse` relation that has the schema (`Husband`, `Wife`) and an `emp` relation that has the schema (`Person`, `Employer`). Appendix A shows an instance of this database and we use this example throughout the paper to illustrate the main ideas of the paper. Data may be missing from the tuples in this database for a number of reasons:

*This work was supported by the Army Research Office under grant number DAAL-03-92-G-0225, by ARPA/Rome Labs contract F30602-93-C-0241 (ARPA Order Nr. A716), by a National Science Foundation Young Investigator Award IRI-9357756 and by a National Science Foundation Grant IRI-9200898

[†]Department of Computer Science, University of Maryland, College Park, Maryland 20742. Email: candan@cs.umd.edu.

[‡]Department of Computer and Information Sciences, Towson State University, Towson, Maryland 21204. Email: grant@midget.towson.edu.

[§]Department of Computer Science, Institute for Advanced Computer Studies & Institute for Systems Research, University of Maryland, College Park, Maryland 20742. Email: vs@cs.umd.edu.

1. **Existential Null:** The person creating the database may know that Elaine has a phone, but s/he may not have the actual number. Hence, a special symbol, denoted *ex_mar* is inserted to denote that a value exists for the `Phone` field of this tuple, but the value is not currently known. The explicit representation is shown as tuple 5 in Appendix A.

2. **Maybe Null:** In the case of some other individual, Ed, the database creator does not know whether or not Ed has a phone. In this case, s/he places a different symbol, denoted *ma_mar* to indicate that a phone number may or may not exist for Ed. This is indicated by tuple 3 in Appendix A.

3. **Place holder Null:** Let us consider the relation `spouse`. In this relation, the individual, Tony, is a bachelor, and hence, the `wife` field is inapplicable to him. Hence, a special symbol, *pl_mar* , called a *placeholder*, is placed in the `wife` field of the tuple associated with Tony. This is shown explicitly as tuple 13 of Appendix A.

4. **Partial Nulls:** If we examine the relation `emp`, it may be the case that the database creator does not remember whether Ed works for IBM or for NCR. S/he knows that Ed works for one of the two, but does not precisely recall which one. In this case, a special kind of value, called a *partial value*, is inserted into the table. This situation is represented by tuple 23 in Appendix A. A partial null may be thought of as an existential null, except that it is somewhat more informative.

5. **Partial Maybe Null:** In the case of Oscar, the database creator may not remember whether Oscar works for NCR or not. For instance, Oscar may be very close to retirement, and the database administrator is not sure whether Oscar has retired (in which case, the field `employer` is inapplicable to Oscar), or Oscar is still an employee, in which case, the value of the field should be NCR. This is an example of a *partial maybe null* where we are not sure whether the field is applicable or not, but if it is applicable, then its value must fall within a specified set. This situation is shown as tuple 26 of Appendix A where the specified set is a singleton.

In this paper, we show how constraints may be used to provide a uniform way of treating the above types of null values. The uniform treatment of different types of null values raises several important problems. In particular, in distributed environments where data gathering is done independently, such problems are very common. For instance, inconsistencies may occur because of the null values – John's `spouse` field may be listed as Sherry in one site, but as a placeholder null in another site. In this case, is John married to Sherry? Or is he a bachelor? The presence of different types of nulls also leads to different ways of coalescing information together. For instance, it may be the case that in one relation, Ed's spouse is listed as an existential null, while in another relation, Ed's spouse is listed as Alice. In this case, with an optimistic approach Ed's spouse's name can be assumed to be Alice, or it may still be assumed to be unknown from a more skeptical point of view. Thus, querying databases with different kinds of null values in them allows us to combine the null values in various ways.

We show how databases containing the above types of null values can be uniformly viewed as databases where each tuple t has an associated constraint C_t. Intuitively, a tuple t may be thought of as being "in" the relation iff C_t is true. Though the idea of using constraints to handle null values in databases is not new (cf. Imielinski-Lipski [13, 14]), the use of constraints to present a *unified way of handling different types of nulls* is, to the best of our knowledge, new and novel. Section 2 shows how databases with constraints can be used to express the semantics of the above types of null values. In Section 3, we develop ways of removing redundant information from null-valued relations and define a "compact" form for such databases. Subsequently, in Section 4, we develop an extension of the relational algebra that handles the above types of nulls. In Section 5, we study properties of this algebra. This consists of establishing various algebraic equalities and inclusions that may be useful for query optimization. In Section 6, we report on the results of experiments that we have conducted on the utility of these algebraic equalities for query optimization.

2 Null-Valued Databases

2.1 Preliminaries

A relation consists of a relational schema (i.e. a list of attributes), and a list of tuples; each tuple is given a tuple-id that is unique for the database under consideration. Thus, a relation may be viewed as a table, whose rows are tuples and whose columns represent attribute values. A given row/tuple t and a given column/attribute A jointly identify a *slot* in the relation table that may be filled in with a *marker*. In the standard relational model of data, each slot must be filled in with a value from the domain of the attribute A being considered. In the case of databases containing null values, however, such as those shown in Appendix A, the slots in relations may be filled in with markers of any one of the following types:

1. (Classical Case) a *value marker* which is any element from the domain of the attribute A,
2. the *existential null marker* ex_mar ,
3. the *maybe null marker*, ma_mar ,
4. the *placeholder null marker*, pl_mar ,
5. a *partial null marker* of the form $pa\_mar = V_s$ where V_s is a nonempty subset of the domain of the attribute A,
6. a *partial maybe marker* of the form $pm\_mar = V_s$ where V_s is a nonempty subset of the domain of the attribute A.

Let va be a function which takes a marker and returns a set of possible values the marker may have. The following table shows the relationship between the markers and va.

X	$va(X)$
va_mar (v)	{v}
ex_mar	D
ma_mar	$D \bigcup \{\perp\}$
pl_mar	$\{\perp\}$
pa_mar (V_s)	V_s
pm_mar (V_s)	$V_s \bigcup \{\perp\}$

Intuitively, $va(ex\_mar) = D$ says that the actual value of an existential marker can be any member of the domain D. Likewise, $va(ma\_mar) = D \bigcup \{\perp\}$ says that the actual value of a maybe marker can be either any member of D, or the symbol $\perp$, denoting a non-existent value. Similarly, $va(pa\_mar\ (V_s)) = V_s$ says that the actual value of a partial null marker of the form pa_mar (V_s) lies in the set V_s.

Tuples that are composed of such markers are called *m-tuples* . Unless specified otherwise, all tuples considered in this paper are *m-tuples* .

2.2 Using Constraints to Handle Null-Valued Databases

In the preceding section, we introduced null valued relations as tables whose slots may be filled in with any of the different types of markers described above. Given any m-tuple t, it is possible to associate with t, a constraint C_t which describes the conditions under which t is "in" the given relation. For instance, if A is an attribute in tuple t with a null marker, M, then tuple t has the associated implicit constraint: $t.A \in va(M)$. This is however not what is reflected in the constraint column as explicit constraints. A typical explicit constraint occurs when a selection is performed on a null-valued tuple like tuple 3 of Appendix A; the associated constraint after a selection can be seen in tuple $2^{\bullet}$ of appendix B.

The syntax of a constraint is similar to that of a first order formula. Let D be a database, R be a relation in D, t be a tuple in R and A be an attribute in the relational schema of R. Then,

- $(D.R.t.A = value)$,
- $(D.R.t.A \neq value)$ and
- $(D.R.t.A \in Set)$

are constraints. Similarly, any boolean combination of these is also a constraint. Note that it is possible to omit database/relation/tupleids if we are not referring to attributes that are in different databases/relations/tuples.

M-tuples that are annotated with constraints of the above type are called *mc-tuples* .

2.3 Null-Valued Relations and Databases

A Null-Valued Relation R consists of two parts:

1. **Relational Schema:** A set of attributes $(R.s)$.
2. **Tuples:** A set of *mc-tuples* . If an *mc-tuple* t is in the relation R, then we write $R \models t$.

In addition to these, we may need to have a subset $(R.k)$ of $R.s$ to behave as a base for the relation. This set will be responsible for determining duplicate information in the relations, i.e. it will behave like a set of key attributes. In this paper we call this set "quasi key attributes". Since their behavior is similar to key attributes, null values may not occur in quasi-key-attributes.

A database containing Null-Valued Relations is called a "Null-Valued Database".

Before defining algebraic operators on Null-Valued Relations, we will first present a formal description of Null-Valued Databases. This provides a formal basis for (later) defining operations on such databases.

2.4 Candidate Tuples

In this subsection, we define the semantics of Null-Valued Databases. We start by describing the meaning of *mc-tuples* .

For the rest of this subsection, let us assume that t_i is an *mc-tuple* in a Null-Valued Relation R, and that $t_i.A$ denotes the marker contained in the attribute A of the *mc-tuple*.

Definition 2.1 (Conforming Tuple) *Let t be a tuple obeying the relational schema $R.s$, and let the domain of each attribute of t be augmented by the new symbol "$\perp$". t is said to be a conforming tuple with respect to t_i (i.e. $t \in conf(t_i)$) iff :*

- *if $t_i.A$ is a va_mar with $va(t_i.A) = \{v\}$, then $(t.A = v)$*
- *if $t_i.A$ is an ex_mar ,then $(t.A \in Dom(A))$ where $Dom(A)$ denotes the domain of attribute A.*
- *if $t_i.A$ is a pa_mar with $va(t_i.A) = V$, then $(t.A \in V)$*
- *if $t_i.A$ is a ma_mar , then $(t.A \in Dom(A)) \vee (t.A = \perp)$ where $Dom(A)$ denotes the domain of attribute A*
- *if $t_i.A$ is a pl_mar , then $(t.A = \perp)$*
- *if $t_i.A$ is a pm_mar with $va(t_i.A) = V \bigcup \{\perp\}$, then $(t.A \in V) \vee (t.A = \perp)$*

Intuitively, a tuple t "conforms" to tuple t_i just in case its attribute values are consistent with the markers in t_i. The set $conf(t_i)$ is the set of all tuples that obey the attribute restrictions of t_i.

Definition 2.2 (Bad Tuple) *Let t be a tuple conforming to the relational schema $R.s$, and suppose the domain of each attribute of t is augmented with a new symbol "$\perp$".*
t is said to be a bad tuple with respect to t_i (i.e. $t \in bad(t_i)$) iff the following holds:

- *t is a conforming tuple with respect to t_i and,*
- *t does not satisfy the associated constraint*

Bad tuples are not equivalent to non-conforming tuples for two reasons. First, in some cases (especially when intermediate tuples are formed as a result of executing some algebraic operations), the constraints associated with tuples may contain references to *other* tuples. This will become apparent later on in this paper. Second, a non-conforming tuple may not be consistent with the markers in the tuple t_i.

Definition 2.3 (Candidate Tuple) *Let t be a tuple obeying the relational schema $R.s$, and suppose the domain of each attribute of t is augmented with a new symbol "$\perp$".*
t is said to be a candidate tuple of t_i (i.e. $t \in can(t_i)$) iff:

1. - *t is a conforming tuple and,*
   - *t satisfies the associated constraint*

or,

2. *There exists a tuple $t' \in bad(t_i)$, and t is t_{null} (t_{null} is a special tuple which denotes a non existent tuple, the use of t_{null} will become clearer when we define Candidate Relations).*

Candidate tuples are those tuples that satisfy both the attributes and the associated constraints of *mc-tuples*; in addition, the null tuple is also a candidate tuple in case condition (2) above holds.

Observe that each *mc-tuple* t_i in a Null-Valued Relation can be described by an expression of the form

$$\Psi(t_i) = \rho(t_i) \wedge \Phi(t_i)$$

where $\rho(t_i)$ describes the part which consists of attribute slots, and $\Phi(t_i)$ is the part which describes the associated constraint.

Example 2.1 *Let t_i be the following mc-tuple:*

tid	*A*	*B*	*C*	*D*	*Constraint*
t_i	*ex_mar*	*3*	*pa_mar* {*3,5*}	*ma_mar*	Θ

then $\rho(t_i) = ((t.A \in Dom(A)) \wedge (t.B = 3) \wedge ((t.C = 3) \vee (t.C = 5)) \wedge ((t.D \in Dom(D) \vee (t.D = \perp))$ *whereas* $\Phi(t_i) = \Theta$.

It is clear from the above definition and example that:

$$t\ satisfies\ (\rho(t_i) \wedge \neg\Phi(t_i)) \longleftrightarrow t \in bad(t_i)$$

and

$$(t\ satisfies\ (\rho(t_i) \wedge \Phi(t_i))) \vee ((\exists t' \in bad(t_i)) \wedge (t = t_{null})) \longleftrightarrow t \in can(t_i)$$

Two *mc-tuples* t_1 and t_2 are considered to be equivalent ($\doteq_t$) iff

$$(can(t_1) = can(t_2)) \wedge (bad(t_1) = bad(t_2))$$

Example 2.2 *Consider mc-tuple 14 in the* `spouse` *relation listed in Appendix A.*

tid	*Husband*	*Wife*	*Constraint*
14	*vic*	*pa_mar* $= \{lisa, joan\}$	*true*

The above tuple is not *equivalent to the tuple below:*

tid	*Husband*	*Wife*	*Constraint*
	vic	*ex_mar*	$(Wife = joan \vee Wife = lisa)$

The original tuple states that "Vic" has a wife and it is also known that his wife is either "Joan" or "Lisa". However the second tuple has a slightly different meaning: "Vic" is known to have a wife, the name of his wife is not known (it could be "Susan" for instance); however, the tuple is in the relation only if the name is either "Joan" or "Lisa". This difference is actually due to the difference in the conforming tuples of these two *mc-tuples*.

Unfortunately, in some cases it may be possible that $\Phi(t_i)$ depends on other tuples, and in such cases it may be impossible to check the truth of $\Phi(t_i)$ by considering only t_i. When we consider those dependent tuples together, we must ensure that $\Phi(t_i)$ holds for all such tuples simultaneously:

Definition 2.4 (Candidate Relation) *Let R be a null valued relation. Then $can(R)$ is constructed by picking a single t from $can(t_i)$ for each mc-tuple t_i in R such that $\bigwedge_{t_i} \Psi(t_i)$ is not falsified. As usual, the null tuple is not shown in $can(R)$.*

Example 2.3 *Let R consist of two mc-tuples .*

R:

tid	Name	Phone-Extension	Constraint
t_1	*john*	$pa\_mar = \{321, 322\}$	*true*
t_2	*john*	$pa\_mar = \{322, 323\}$	*true*

The following is a candidate relation for R:

Name	Phone-Extension
john	321
john	322

In this example, the first tuple was obtained from t_1, while the second was obtained from t_2.

Definition 2.5 (Representation of a Relation) *Let R be a null valued relation. Then the set*

$$Rep(R) = \{R_i | R_i \text{ is a candidate relation of } R\}$$

is called the representation of R.

Two relations R_1 and R_2 are considered to be equivalent ($\doteq_r$) iff

$$Rep(R_1) = Rep(R_2)$$

Again, in some cases, it may be possible that *mc-tuples* in relation R_1 depend on *mc-tuples* in R_2. When two such relations are considered together, then the truth of $\bigwedge_{R_j} \bigwedge_{t_i} \Psi(t_i)$ must be guaranteed.

Example 2.4 *If we consider the relation in the previous example, $Rep(R)$ is $\{R_1, R_2, R_3, R_4\}$ where:*

R_1:

Name	Phone-Extension
john	321
john	322

R_2:

Name	Phone-Extension
john	322

R_3:

Name	Phone-Extension
john	321
john	323

R_4:

Name	Phone-Extension
john	322
john	323

3 Redundancy Removal in Null-Valued Databases

A relation may contain redundancy due to the existence of tuples that duplicate information. When nulls are not allowed in tuples the concept of "duplication" of information is clear – two tuples are duplicates of one another iff they are identical. Hence, eliminating such duplicates is done more or less automatically in the standard relational model of data. However, when nulls are present, we have to be more careful in defining the conditions that cause duplication of information. Suppose t_1 and t_2 are two *mc-tuples* (with associated constraints C_1, C_2 respectively). Clearly, t_1 and t_2 should have some commonality if they have the same values for the quasi-key attributes. In this case, we look at each marker occurring in $t_1 =< a_1, \ldots, a_n, C_1 >$ and/or in $t_2 =< b_1, \ldots, b_n, C_2 >$ and attempt to combine each a_i with the corresponding b_i to yield a "composite" *mc-tuple* $t_3 =< d_1, \ldots, d_n, (C_1 \wedge C_2) >$ which is jointly entailed by both t_1 and t_2 under the constraint $(C_1 \wedge C_2)$.

Similarly, a relation containing null values and constraints may also include various inconsistencies due to incomplete information. To see this consider the following example:

Example 3.1 *Assume that there are two sensors measuring the speed of a particle in a laboratory experiment. The first sensor reads a velocity of* $2.102 - 2.105 \times 10^8 m/sec$ *whereas the second sensor reads a velocity of* $2.103 - 2.107 \times 10^8 m/sec$. *What is the speed of the particle ?*

To reason about databases containing such redundancies and inconsistencies, we must have some tools to remove information overlap, redundancies and inconsistencies. There are various methods that may be used in building such tools. In this section we investigate some of these methods.

3.1 Overlap Removal

Two *mc-tuples* in a relation R are said to overlap if they have the same quasi-keys. This basically means that there is redundancy in the relation if the intersection of their candidate tuple sets is different from both $\emptyset$ and $\{t_{null}\}$; and/or there is an inconsistency in the relation if the difference of their candidate tuple sets is different from both $\emptyset$ and $\{t_{null}\}$. Both of these problems can be removed using one of the strategies listed below:

Let t_i and t_j be two *mc-tuples* such that

$$(\forall_{quasi-key\ attributes\ K})((t_i.K = va\_mar\ ) \wedge (t_j.K = va\_mar\ ) \wedge (va(t_i.K) = va(t_j.K)))$$

and let $\Phi(t_i) = \Phi(t_j)$.

1. **(Strategy 1)** Remove t_i and t_j from the relation and add the *mc-tuple* t_l such that $\Phi(t_l) = \Phi(t_i)$ and
$$conf(t_l) = conf(t_i) \cap conf(t_j)$$

2. **(Strategy 2)** Remove t_i and t_j from the relation and add the *mc-tuple* t_l such that $\Phi(t_l) = \Phi(t_i)$ and
$$((conf(t_i) \cap conf(t_j) \neq \emptyset) \rightarrow (conf(t_l) = (conf(t_i) \cap conf(t_j))) \wedge$$
$$((conf(t_i) \cap conf(t_j) = \emptyset) \rightarrow (conf(t_l) = (conf(t_i) \cup conf(t_j)))$$

3. **(Strategy 3)** Remove t_i and t_j from the relation and add the *mc-tuple* t_l such that $\Phi(t_l) = \Phi(t_i)$ and
$$conf(t_l) = conf(t_i) \cup conf(t_j)$$

We will now show how the above strategies can be used to build tools for removing redundancies/inconsistencies (merging) in Null-Valued relations. Before doing so, we need to describe a method to compose various kinds of null markers and specify:

1. a precise definition of what it means to compose markers and
2. precise conditions under which two tuples may be "merged".

the next subsection answers to (1) above. Section 3.1.1 deals with the first point above, while Section 3.1.2 deals with the second.

3.1.1 Marker-Composition

Marker-Composition is a binary operation on markers that assesses the information contained in its operands, and attempts to compose them together. In order to keep the intuitiveness of the operation, the marker composition operator, $\odot_{mc}$, should satisfy the idempotent, associative and commutative properties:

1. $X \odot_{mc} X = X$
2. $X \odot_{mc} (Y \odot_{mc} Z) = (X \odot_{mc} Y) \odot_{mc} Z$
3. $X \odot_{mc} Y = Y \odot_{mc} X$

There are various ways to define the marker composition operator. We discuss some alternative marker composition methods below.

Suppose *mc-tuples* t_1, t_2, with identical values for the quasi-key attributes, are in the same relation, and suppose that for attribute A, they have markers n_1, n_2 respectively in that attribute slot. According to *mc-tuple* t_1, the value of the attribute A is in $va(n_1)$ while according to t_2, the value of the attribute A is in $va(n_2)$.

1. **First Marker Composition Strategy:** As *mc-tuples* jointly indicate that the value of the attribute A is in the intersection of $va(n_1)$ and $va(n_2)$, we may take the set of possible values as $va(n_1) \bigcap va(n_2)$ If this intersection results in a nonempty set then there is a marker n whose domain corresponds precisely to the intersection of the domains of n_1 and n_2, and this marker n is the result of the marker composition of n_1, n_2. If not, we remove the corresponding *mc-tuples* from the relation, because they represent an inconsistency. This marker composition strategy is depicted in Table 1.

1st operand	2nd operand	Result
va_mar	va_mar	va_mar {or - }
ex_mar	ex_mar	ex_mar
ex_mar	ma_mar	ex_mar
ex_mar	pa_mar	pa_mar
ex_mar	pl_mar	-
ex_mar	pm_mar	pa_mar
ex_mar	va_mar	va_mar
ma_mar	ma_mar	ma_mar
ma_mar	pa_mar	pa_mar
ma_mar	pl_mar	pl_mar
ma_mar	pm_mar	pm_mar
ma_mar	va_mar	va_mar
pl_mar	pl_mar	pl_mar
pl_mar	pa_mar	-
pl_mar	pm_mar	pl_mar
pl_mar	va_mar	-
pa_mar	pa_mar	pa_mar {or - }
pa_mar	va_mar	va_mar {or - }
pm_mar	va_mar	va_mar {or - }
pm_mar	pa_mar	pm_mar {or - }
pm_mar	pm_mar	pm_mar {or pl_mar }

TABLE 1: First Marker Composition Strategy

2. **Second Marker Composition Strategy:** In the preceding strategy the resulting domain may be empty. At this stage, one may wish to take the union of $va(n_1)$ and $va(n_2)$. The second marker composition strategy therefore is $va(n_1) \bigcap va(n_2)$ – however, if that intersection is empty, then the composition is $va(n_1) \bigcup va(n_2)$. The computation of the union reflects the fact that the actual value is in the domain of either n_1 or n_2, but as there is an inconsistency involved, the system simply returns all the possibilities. We would like this marker composition method to conform to the strategies described at the beginning of section 3.1. If this marker composition strategy is applied to *mc-tuples* t_1 and t_2, then even if $va(n_1) \bigcap va(n_2) = \emptyset$ for just a single attribute in the relational schema, the composition of t_1 and t_2 is performed by taking the unions of domains of "each and every attribute". This marker composition strategy is depicted in Table 2.

1st operand	2nd operand	Result
va_mar	va_mar	va_mar {or pa_mar }
ex_mar	ex_mar	ex_mar
ex_mar	ma_mar	ex_mar
ex_mar	pa_mar	pa_mar
ex_mar	pl_mar	ma_mar
ex_mar	pm_mar	pa_mar
ex_mar	va_mar	va_mar
ma_mar	ma_mar	ma_mar
ma_mar	pa_mar	pa_mar
ma_mar	pl_mar	pl_mar
ma_mar	pm_mar	pm_mar
ma_mar	va_mar	va_mar
pl_mar	pl_mar	pl_mar
pl_mar	pa_mar	pm_mar
pl_mar	pm_mar	pm_mar
pl_mar	va_mar	pm_mar
pa_mar	pa_mar	pa_mar
pa_mar	va_mar	va_mar {or pa_mar }
pm_mar	va_mar	va_mar {or pm_mar }
pm_mar	pa_mar	pa_mar {or pm_mar }
pm_mar	pm_mar	pl_mar {or pm_mar }

TABLE 2: Second Marker Composition Strategy

3. **Third Marker Composition Strategy:** The third marker composition strategy takes the composition to be $va(n_1) \bigcup va(n_2)$. This marker composition strategy is depicted in Table 3.

1st operand	2nd operand	Result
va_mar	va_mar	va_mar {or pa_mar }
ex_mar	ex_mar	ex_mar
ex_mar	ma_mar	ma_mar
ex_mar	pa_mar	ex_mar
ex_mar	pl_mar	ma_mar
ex_mar	pm_mar	ma_mar
ex_mar	va_mar	ex_mar
ma_mar	ma_mar	ma_mar
ma_mar	pa_mar	ma_mar
ma_mar	pl_mar	ma_mar
ma_mar	pm_mar	ma_mar
ma_mar	va_mar	ma_mar
pl_mar	pl_mar	pl_mar
pl_mar	pa_mar	pm_mar
pl_mar	pm_mar	pm_mar
pl_mar	va_mar	pm_mar
pa_mar	pa_mar	pa_mar
pa_mar	va_mar	pa_mar
pm_mar	va_mar	pm_mar
pm_mar	pa_mar	pm_mar
pm_mar	pm_mar	pm_mar

TABLE 3: Third Marker Composition Strategy

As an example of how these tables are constructed, let us consider Table 2 above. Consider the first row in this table. If the same value occurs in both attribute slots, then there is no inconsistency – we take that value itself – however, if the values are different, then the result is a *pa_mar* marker denoting that there is an inconsistency – in this case, the two values involved in the inconsistency are returned as output. In general, all the conflicts in this table are resolved similarly.

Properties Reconsidered: In order to evaluate the three natural strategies articulated above, we need to check if they satisfy the idempotent, associative and commutative properties. In the case of the first and the third strategies, it is easy to see that these properties hold by the properties of set intersection and union respectively, but the second strategy does not satisfy the associativity requirement. To see this, observe that

$$(ex\_mar \odot_{mc} ex\_mar) \odot_{mc} pl\_mar = ex\_mar \odot_{mc} pl\_mar = ma\_mar .$$

However,

$$ex\_mar \odot_{mc} (ex\_mar \odot_{mc} pl\_mar) = ex\_mar \odot_{mc} ma\_mar = ex\_mar .$$

The user should choose which marker composition strategy s/he wishes to adopt, keeping in mind the different semantics of these approaches.

3.1.2 When to Merge

Even if an initial set of relations does not contain overlaps, when executing a query involving various relational operators, the interim tables that are constructed could contain various kinds of overlaps. For instance, union may cause redundant *mc-tuples* from different relations to come together. Hence, we need an operator to minimize the redundancy in these relations (including intermediate relations) to keep our database free of overlaps.

In this section, we explain how to use marker composition methods to merge *mc-tuples* that contain redundant information. Two *mc-tuples* t_1 and t_2 (with associated constraints C_1, C_2 respectively) may be merged if either of the two situations below occurs:

1. **Quasi-Key Overlaps:** t_1 and t_2 are said to be quasi-key overlapping if for each quasi-key attribute slot, the values of t_1 and t_2 are identical, and C_1 is not logically equivalent to $\neg C_2$.

2. **Value Overlaps:** t_1 and t_2 are said to be value overlapping if for every attribute-slot, t_1 and t_2 have identical markers.

If we look at the table below, where the quasi-key attribute is `husband`, then *mc-tuples* $t_1^\bullet, t_2^\bullet, t_3^\bullet, t_4^\bullet$ are all quasi-key overlapping, while *mc-tuples* $t_2^\bullet$ and $t_3^\bullet$ are value overlapping.

tid	Husband	Wife	Employer	Constraint
$t_1^\bullet$	john	*ex_mar*	ncr	$(Wife = lisa)$
$t_2^\bullet$	john	*ex_mar*	ibm	$(Wife = lisa)$
$t_3^\bullet$	john	*ex_mar*	ibm	$(Wife = joan)$
$t_4^\bullet$	john	audrey	*ex_mar*	$(Employer = ncr)$.

TABLE 4: Overlap Examples

When overlaps of the above kind occur, we would like to *minimize the overlap*. Such a minimization serves many purposes. First, *mc-tuples* with the same quasi-keys, but with different non-quasi-key attribute markers are merged into a single *mc-tuple* so that the information in the non-key attributes may be "amalgamated." Second, many *mc-tuples* with identical attribute markers, but different constraints get grouped together into a single, unified *mc-tuple*. Third, after the minimization, different *mc-tuples* associated with the same quasi-key attributes have mutually conflicting constraints. We now show how to remove the above types of overlaps:

Removing Quasi-Key Overlaps: Suppose t_1 is the *mc-tuple* $< a_{1,1} ... a_{1,n}, C_1 >$ and t_2 is the *mc-tuple* $< a_{2,1} ... a_{2,n}, C_2 >$, and suppose these *mc-tuples* are quasi-key overlapping but not value overlapping. Then we introduce a new *quasi-key-merged mc-tuple*:

$$t_1 \odot_{q,k} t_2 = t_3' =< a_{3,1} ... a_{3,n}, (C_1 \wedge C_2) >$$

where $a_{3,i} = a_{1,i} \odot_{mc} a_{2,i}$. "$\odot_{q,k}$" is the quasi-key-overlap removal operator; here 'k' denotes the set of quasi-keys, while $\odot_q$denotes the quasi-key removal operator.

In addition to the above operation, we also replace t_1 and t_2 by the new *mc-tuples* :

$$t_1' =< a_{1,1} ... a_{1,n}, (C_1 \wedge \neg C_2) >$$

and

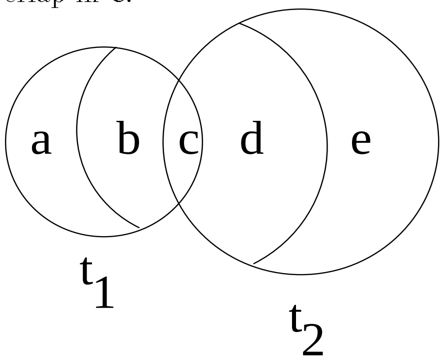
$$t_2' =< a_{2,1} ... a_{2,n}, (C_2 \wedge \neg C_1) >$$

unless C_1 is logically equivalent to C_2, in which case, we omit these two *mc-tuples*because the constraint is equivalent to false.

This operation can be visualized as follows : Let t_1 be composed of three parts: **a**, **b** and **c**; similarly let t_2 be composed of **c**, **d** and **e** where

- **a** satisfies $= \rho(t_1) \wedge (\Phi(t_1) \wedge \neg\Phi(t_2))$
- **e** satisfies $= \rho(t_2) \wedge (\Phi(t_2) \wedge \neg\Phi(t_1))$
- **b** satisfies $= (\rho(t_1) \wedge \neg\rho(t_2)) \wedge (\Phi(t_1) \wedge \Phi(t_2))$
- **d** satisfies $= (\rho(t_2) \wedge \neg\rho(t_1)) \wedge (\Phi(t_1) \wedge \Phi(t_2))$
- **c** satisfies $= \rho(t_1) \wedge \rho(t_2) \wedge (\Phi(t_1) \wedge \Phi(t_2))$

Note that these two *mc-tuples* overlap in **c**:



After the removal of the overlap we have the following:

- t_1' is **a**,
- t_2' is **e**,
- t_3 is either **c** or **b** $\bigcup$ **c** $\bigcup$ **d** depending on the chosen semantics.

For example, consider *mc-tuples* $t_1^\bullet$ and $t_2^\bullet$ in table 4. The quasi-key-merge of these two *mc-tuples* is (assuming a union based marker composition strategy):

tid	Husband	Wife	Employer	Constraint
t	john	*ex_mar*	*pa_mar* $= \{ibm, ncr\}$	$(Wife = lisa)$

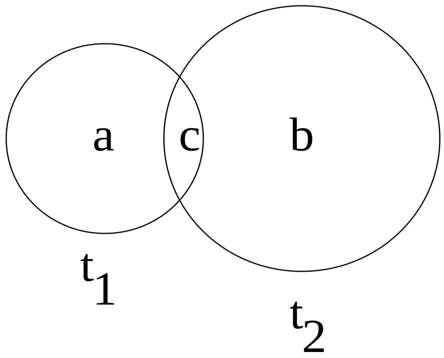
Removing Value Overlaps: Suppose $t_1 =< e_1, ..., e_n, C_1 >$ and $t_2 =< e_1, ..., e_n, C_2 >$ are value-overlapping *mc-tuples* . The *value-merge* of these two *mc-tuples* is:

$$t_1 \odot_v t_2 = t_3 =< e_1, ..., e_n, (C1 \vee C_2) >.$$

This operation can be visualized as follows : Let t_1 be composed of two parts: **a** and **c**; similarly let t_2 be composed of **b**, and **c** where:

- **a** satisfies $= \rho(t_1) \wedge (\Phi(t_1) \wedge \neg\Phi(t_2))$
- **b** satisfies $= \rho(t_1) \wedge (\Phi(t_2) \wedge \neg\Phi(t_1))$
- **c** satisfies $= \rho(t_1) \wedge \Phi(t_1) \wedge \Phi(t_2)$

Note that $\rho(t_1) = \rho(t_2)$.



The result of the operation will be an *mc-tuple* t_3 whose candidate set is $\mathbf{a} \bigcup \mathbf{b} \bigcup \mathbf{c}$.

For example, consider *mc-tuples* $t_2^\bullet$ and $t_3^\bullet$ in table 4. The value-merge of these two *mc-tuples* is:

tid	Husband	Wife	Employer	Constraint
t	john	*ex_mar*	ibm	$((Wife = lisa) \vee (Wife = joan))$

3.1.3 Canonical Forms

Having defined different types of overlaps, as well as different strategies to remove overlaps, we may now define a *canonical form* for tuples/relations/databases. Intuitively, we would like relations in this canonical form to have no overlaps and to be as simple as possible.

We now define the notion of "canonical forms" of relations. Such "canonical forms" minimize redundancy in tuples.

Definition 3.1 CFR-Canonical Form for Relations: *A relation is said to be in* **CFR** *if it does not contain two mc-tuples that are either value-overlapping or quasi-key overlapping. Relations in* **CFR** *are called* **Canonical Relations**.

A direct extension of this definition for databases is:

Definition 3.2 CFD-Canonical Form for Databases: *A database is said to be in* **CFD** *if all relations in it are in* **CFR**. *Databases in* **CFD** *are going to be called, for short, Canonical Databases.*

In the next subsection, we define a special operator, denoted as $\mathcal{C}$, that takes a given relation R and a set K of quasi-key attributes as input, and returns as output, a canonical version of that relation by getting rid of the redundant and overlapping information.

3.2 Compaction Operator

We define the Compaction Operator ($\mathcal{C}$) as follows:

Let I and O be two sets of *mc-tuples* such that I contains the same *mc-tuples* as the input relation and O is empty.

1. Choose two *mc-tuples* t_1 and t_2 from I and set I to $I - \{t_1, t_2\}$.
2. If t_1 and t_2 are not **overlapping** then put them in O.
3. Else if t_1 and t_2 are **value-overlapping** then apply the value-overlap removal procedure to t_1 and t_2 and put the resulting *mc-tuple* in O,
4. Else if t_1 and t_2 are **quasi-key-overlapping** then apply the quasi-key-overlap removal procedure to t_1 and t_2 and put the resulting *mc-tuples* in O (note that the resulting *mc-tuples* have disjoint candidate tuple sets).
5. While there are *mc-tuples* in I do
   (a) Choose an *mc-tuple* t from I and set I to $I - \{t\}$.
   (b) Compare t with each *mc-tuple* in O
   (c) If t has a **quasi-key-overlap** with an *mc-tuple* t' in O then
      i. Remove t' from O
      ii. Apply the quasi-key-overlap removal procedure to t and t'
      iii. Put the quasi-key-merged *mc-tuple* in O and put the rest of them in I
   (d) Else if t has a **value-overlap** with an *mc-tuple* t' in O then
      i. Remove t' from O
      ii. Apply the value-overlap overlap removal procedure to t and t'
      iii. Put the resulting *mc-tuple* in O.
   (e) Else put t in O.
6. At this stage I is empty, and the *mc-tuples* in O contains no overlap.

Note that at each stage we guarantee that the *mc-tuples* in O have pairwise disjoint candidate tuple sets.

3.2.1 Correctness

The above algorithm obeys the definition of redundancy removal semantics described at the beginning of this section. Note however that $Rep(\mathcal{C}(R))$ may be different from $\mathcal{C}(Rep(R))$ – in the latter, the compaction operator is applied to each relation in $Rep(R)$. However, if the first strategy is used, then these two computations yield the same solution.

The following example shows that $Rep(\mathcal{C}(R))$ may be different from $\mathcal{C}(Rep(R))$ if the union based marker composition strategy is used.

Example 3.2 *Suppose relation R consists of two mc-tuples .*

R:

tid	Name	Phone-Extension	Constraint
t_1	*john*	$pa\_mar = \{321, 322\}$	*true*
t_2	*john*	$pa\_mar = \{322, 323\}$	*true*

Then if the union based compaction is applied:

$\mathcal{C}(R)$:

tid	Name	Phone-Extension	Constraint
tc	*john*	$pa\_mar = \{321, 322, 323\}$	*true*

Rep(R) is $\{R_1, R_2, R_3, R_4\}$ *where:*

R_1:

Name	Phone-Extension
john	321
john	322

R_2:

Name	Phone-Extension
john	322

R_3:

Name	Phone-Extension
john	321
john	323

R_4:

Name	Phone-Extension
john	322
john	323

$\mathcal{C}(R_1)$:

tid	Name	Phone-Extension	Constraint
t'_1	*john*	$pa\_mar = \{321, 322\}$	*true*

$\mathcal{C}(R_2)$:

tid	Name	Phone-Extension	Constraint
t'_1	*john*	322	*true*

$\mathcal{C}(R_3)$:

tid	Name	Phone-Extension	Constraint
t'_1	*john*	$pa\_mar = \{321, 323\}$	*true*

$\mathcal{C}(R_4)$:

tid	Name	Phone-Extension	Constraint
t'_1	*john*	$pa\_mar = \{322, 323\}$	*true*

However, $Rep(\mathcal{C}(R))$*is:*

R'_1:

Name	Phone-Extension
john	321

R'_2:

Name	Phone-Extension
john	322

R'_3:

Name	Phone-Extension
john	323

4 Null-Valued Algebraic Operations

We are now at a stage where we can define various algebraic operators to be used with null-valued databases. In this section, we define an algebra that extends the standard relational algebra to handle databases that contain the different kinds of null values discussed so far.

The algebraic operators can be defined using the formalism introduced earlier. This approach provides a better understanding of the way the algebraic operators should behave under the existence of null values. The following items provide definitions of the algebraic operators :

- **Selection:**

$$t_j \in \sigma_\Theta(R) \longleftrightarrow (\exists t_i \in R)((\rho(t_j) = \rho(t_i)) \wedge (\Phi(t_j) \equiv \Phi(t_i) \wedge \Theta))$$

- **Projection:**

$$t_j \in \Pi_A(R) \longleftrightarrow (\exists t_i \in R)((\rho(t_j) = \rho'(t_i)) \wedge (\Phi(t_j) = \Phi(t_i))$$

Note that $\rho'(t_i) = \rho(t_i)$ except that the parts which include any of the attributes not in A are omitted.

- **Union:**

$$t_j \in (R_1 \bigcup R_2) \longleftrightarrow (\exists t_i \in R_1)(t_i \doteq t_j) \vee (\exists t_i \in R_2)(t_i \doteq t_j)$$

- **Intersection:**

$$t_j \in (R_1 \bigcap R_2) \longleftrightarrow (\exists t_i \in R_1)(\exists t_k \in R_2)((\rho(t_j) = \rho(t_i) \wedge \rho(t_k)) \wedge (\Phi(t_j) = \Phi(t_i) \wedge \Phi(t_k)))$$

- **Difference:**

$$\begin{aligned} t_j \in (R_1 Dif R_2) \longleftrightarrow (\exists t_i \in R_1)(\exists t_k \in R_2)(((\rho(t_j) = \rho(t_i) \wedge \neg\rho(t_k)) \wedge (\Phi(t_j) = \Phi(t_i)))\vee \\ ((\rho(t_j) = \rho(t_i)) \wedge (\Phi(t_j) = \Phi(t_i) \wedge \neg\Phi(t_k)))) \end{aligned}$$

- **Join:**

$$\begin{aligned} t_j \in (R_1 \bowtie R_2) \longleftrightarrow & (\exists t_i \in R_1)(\exists t_k \in R_2) \\ & ((\forall_{joining\ attributes\ A})((t_i.A\ is\ a\ va\_mar\ ) \wedge\ (t_k.A\ is\ a\ va\_mar\ )\wedge \\ & \qquad (t_i.A = t_k.A))\wedge \\ & \qquad ((\rho(t_j) = \rho(t_i) \wedge \rho(t_k)) \wedge (\Phi(t_j) = \Phi(t_i) \wedge \Phi(t_k)))) \end{aligned}$$

The above definitions extend the algebraic operators on Conditional Tables given by Imielinski and Lipski [13, 14] because the operators in [14] apply only to databases containing existential nulls – our operators apply to all the types of nulls described at the beginning of this paper. Though all the types of nulls described in this paper are discussed in [3], they do not provide a *unified* framework of handling these nulls jointly, which we do here.

After giving the formal definitions of the algebraic operators, in the following subsections, we show how to build such operators.

4.1 Selection

Definition 4.1 (Selection Condition) *A selection condition is any boolean expression constructed using the attributes in the relational schema.*

For instance in the case of the `spouse` relation, the condition (`Wife` = `joan`) is a selection condition.

When null values/constraints are present in the database, the evaluation of selection conditions is non-trivial. There are two main cases that a selection operator must take into account when operating in an environment containing null values. These cases are:

1. A value marker appears for an attribute in the selection condition – in this case, as in the classical relational database scenario, we simply substitute the value encapsulated in the marker into the formula by instantiating the appropriate variable to this value.

2. A null marker appears for an attribute in the selection condition. This case is complicated because there may be a number of options on how to instantiate the variables in the selection criterion.

In the next subsections, we describe how to handle the null markers in the attributes.

4.1.1 Value Markers in the Selection Condition

When the selection operator finds a value marker for one of the attributes specified in the selection condition, it must instantiate the value encapsulated in the marker to the corresponding variable. Obviously if the selection condition consists only of value markers, then at the end, the selection condition will reduce to a form in which no variables (attributes) appear. At this time, it is possible to check the condition for satisfiability. If it is satisfied, then the corresponding *mc-tuple* is placed in the result, otherwise it is omitted. Traditional databases, where there are no nulls, correspond to this case where all *mc-tuples* only contain value markers.

4.1.2 Null Markers in the Selection Condition

If we are attempting to determine whether a given *mc-tuple* t satisfies the selection condition, and if *mc-tuple* t contains null values in it, then we need to determine ways of evaluating these nulls.

1. **Single Null Marker in the Selection Condition**

   Let us first assume that the *mc-tuple* t has only one null value in it and that this null value occurs in attribute X. Let Φ be the selection condition that contains X. Our procedure will consider *mc-tuple* t and either return nothing, or return a *mc-tuple* t' as the output of the selection condition. t' may be constructed from t in a precise way, as shown in the table below.

Input Tuple (t)		Output Tuple(t')	
Marker in X	Selection Condition	Marker in X	Constraint
ex_mar	Φ	*ex_mar*	$\Phi(X)$
ma_mar	Φ	*ma_mar*	$\Phi(X)$
pl_mar	Φ	—	—
pa_mar (S)	Φ	*pa_mar* (S)	$\Phi(X)$
pm_mar (S)	Φ	*pm_mar* (S)	$\Phi(X)$

   The third row of the above table indicates that when the selection condition contains a *pl_mar* then we do not get an *mc-tuple* in the result. Remembering that *pl_mar* means there is no value for the corresponding attribute, it is easy to see why this is so. When the constraint depends on a *pl_mar* the constraint can not be satisfied. So, the *mc-tuple* is not in the resulting relation. Similarly if none of the members of a *pa_mar* or *pm_mar* satisfy the selection condition, then the corresponding tuple is omitted from the result.

   As an example, consider the following query on the relation `emp` defined in Appendix A:

   SELECT NAME,EMPLOYER
   FROM emp
   WHERE EMPLOYER = ncr.

In this case:

(a) *Mc*-tuple 22 is clearly in the answer.

(b) On the other hand, consider *mc-tuple* 23. This says Ed works for either IBM or NCR, but we don't know which. According to the above table, we return the tuple:

tid	Person	Employer	Constraint
$23^{\bullet}$	ed	$pa\_mar = \{ibm, ncr\}$	$(Employer = ncr)$

This *mc-tuple* says that the *mc-tuple* $23^{\bullet}$ is in the relation only if the employer field is equal to NCR (which it is not known to be right now).

(c) Consider *mc-tuple* 25 which has a maybe marker in it. In this case, the tuple should be in the result of this select query just in case Vic's employer does exist and is NCR. As a consequence, we modify *mc-tuple* 25 to *mc-tuple* $25^{\bullet}$ by including this condition, and place the result in the relation.

tid	Person	Employer	Constraint
$25^{\bullet}$	vic	ma_mar	$(Employer = ncr)$

2. **Multiple Null Markers in the Selection Condition**

The next question is: what do we do if the selection condition requires that we consider an *mc-tuple* t that has multiple null values in it? In this case, we must apply the methods described in the preceding section to every null marker in t, and then merge the constraints resulting from this process. For instance, let us consider a complex selection condition that involves arithmetic operations such as: $(\Phi\ :\ X = 2 \times Y \times Z)$. where X, Y and Z are attributes. Let R be a relation with the schema $R < X, Y, Z >$, and let t be the following *mc-tuple*:

$$< va\_mar\ (365.00),\ ma\_mar\ ,\ ex\_mar\ , true >$$

in R. When the selection condition is applied to t, the resulting *mc-tuple* is:

$$< va\_mar\ (365.00),\ ma\_mar\ ,\ ex\_mar\ , \Phi(365.00, Y, Z) > .$$

If instead, t were of the form:

$$< va\_mar\ (365.00),\ ma\_mar\ ,\ ex\_mar\ , C >$$

where C is a constraint over *mc-tuple* t, then the resulting tuple is:

$$< va\_mar\ (365.00),\ ma\_mar\ ,\ ex\_mar\ , C \wedge \Phi(365.00, Y, Z) > .$$

Note that the constraint part of this *mc-tuple* has been modified by appending a new conjunct to the original constraint.

4.1.3 Selection Predicate

A Selection Predicate is a ternary predicate of the form:

$$\Lambda(\Theta, t_1, t_2).$$

This atom is true iff $t_2 = \sigma_\Theta(t_1)$, i.e. iff at least one of the following two conditions hold:

1. When the process described in subsections 4.1.1 and 4.1.2 is applied to t_1 and the selection condition Θ we get:
   - The resulting modified selection condition is Θ',
   - All the variables in Θ' are instantiated with values,
   - Θ' evaluates to "true",
   - t_2 is equal to t_1.
2. When the process described in subsections 4.1.1 and 4.1.2 is applied to t_1 and the selection condition Θ, we get:
   - The resulting modified selection condition is Θ',
   - Θ' includes uninstantiated variables,
   - t_2 is equal to t_1 with the modified constraint.

4.1.4 Selection

This operator (denoted by σ) selects the *mc-tuples* satisfying the selection condition Θ and returns them in the output relation.

$$a \; \epsilon \; \sigma \; (R, \Theta) \longleftrightarrow in\_select(a, R, \Theta)$$

where *in_select* is defined as follows:

$$in\_select(x, r, \Theta) \leftarrow (r \models y) \wedge \; \Lambda(\Theta, y, x)$$

Example 4.1 *For example, consider the* `emp` *relation in Appendix A, and consider the query*

*SELECT **
FROM emp
WHERE Employer = ncr.

In this case, the following mc-tuples are returned:

tid	*Person*	*Employer*	*Constraint*
$22^\bullet$	*sherry*	*ncr*	*true*
$23^\bullet$	*ed*	$pa\_mar = \{ibm, ncr\}$	$(Employer = ncr)$
$25^\bullet$	*vic*	ex_mar	$(Employer = ncr)$
$26^\bullet$	*oscar*	$pa\_mar = \{ncr\}$	$(Employer = ncr)$

4.1.5 Correctness

The selection operator is correct in the sense that $\sigma(Rep(R)) = Rep(\sigma(R))$. To see this note that:
$\sigma(Rep(R)) = \{\sigma(R')|R' = can(R)\}$ and,
$Rep(\sigma(R)) = \{R'|R' = can(\sigma(R))\}$.

Let R' be a candidate relation of R. If tuple t' is in R' and if it also satisfies the selection condition, then it will be in $\sigma(R')$.

There is an *mc-tuple* t in R such that t' satisfies

$$\Psi(t) = \rho(t) \wedge \Phi(t).$$

Since t' also satisfies Θ, we can conclude that t' satisfies

$$\Psi(t) \wedge \Theta = \rho(t) \wedge \Phi(t) \wedge \Theta.$$

However, if we look at the definition of the selection operator, we can see that if an *mc-tuple* t'' is in $\sigma(R)$ then

$$\Psi(t'') = \rho(t'') \wedge \Phi(t'') = \rho(t''') \wedge \Phi(t''') \wedge \Theta$$

where t''' is a *mc-tuple* in relation R.

Hence for each tuple t' in $\sigma(R')$, there exists an *mc-tuple* t'' in $\sigma(R)$, and for each *mc-tuple* t'' in $\sigma(R)$, there is a tuple t' in R'. This basically means that $\sigma(R')$ is a candidate relation of $\sigma(R)$ which proves the above claim.

Example 4.2 *Suppose we have the following relation* (R)*:*

tid	*X*	*Constraint*
1	*ex_mar*	*true*
2	*5*	*true*

Let us also assume that a selection operation is performed on this relation with the selection condition $\Theta(X) = ((X = 5) \vee (X = 7))$.

By the above definition $\sigma_\Theta(R)$ *is :*

tid	*X*	*Constraint*
$1'$	*ex_mar*	$((X = 5) \vee (X = 7))$
$2'$	*5*	$((X = 5) \vee (X = 7))$

$Rep(\sigma_\Theta(R))$ is $\{R_1,R_2\}$ such that, $R_1 = \begin{array}{|c|}\hline X \\ \hline 5 \\ \hline\end{array}$ *and* $R_2 = \begin{array}{|c|}\hline X \\ \hline 7 \\ \hline 5 \\ \hline\end{array}$.

On the other hand, $Rep(R)$ is $\{R'_1,R'_2,\ldots\}$ such that, $R'_1 = \begin{array}{|c|}\hline X \\ \hline 1 \\ \hline 5 \\ \hline\end{array}$, $R'_2 = \begin{array}{|c|}\hline X \\ \hline 2 \\ \hline 5 \\ \hline\end{array}$ .... *Hence,* $\sigma_\Theta(Rep(R))$

is $\{R_1'', R_2''\}$ *such that,* $R_1'' = \begin{array}{|c|}\hline X \\ \hline 5 \\ \hline\end{array}$, $R_2'' = \begin{array}{|c|}\hline X \\ \hline 7 \\ \hline 5 \\ \hline\end{array}$.

Thus $\sigma_\Theta(Rep(R)) = Rep(\sigma_\Theta(R))$.

Correctness of the other relational operators follow similarly.

4.2 Projection

The projection operator is easy to define, because its behavior is very similar to the behavior of the standard projection operator. The main difference is that the constraints associated with some of the *mc-tuples* may undergo a change when this operation is executed.

For example, consider the query

PROJECT Person
FROM (SELECT * FROM emp WHERE Employer = ncr)

The table returned by the inner select is shown in Example 4.1. If we were to do a standard project on this (and naively carry the constraints along, as before, by replacing old tuple-ids by the new tuple-ids), we would obtain the table:

tid	Person	Constraint
22°	sherry	true
23°	ed	$(Employer = ncr)$
25°	vic	$(Employer = ncr)$
26°	oscar	$(Employer = ncr)$

However, these expressions refer to the *non-existent* field, `employer`, which makes the future evaluation of these constraints impossible. Nevertheless, the `Person`, Ed, is in the projected name field iff the constraint associated with the *original mc-tuple* was satisfied. Hence, we need to allow the constraints in a projection to refer to the fields of *other mc-tuples*. In this example, this would be done as follows:

tid	Person	Constraint
22°	sherry	true
23°	ed	$(23^\bullet.Employer = ncr)$
25°	vic	$(24^\bullet.Employer = ncr)$
26°	oscar	$(25^\bullet.Employer = ncr)$

Addition of tuple pointers necessitates the storage of the intermediate tables. If an intermediate table is deleted, those *mc-tuples* which point to it need to be reconsidered. The associated constraints of those *mc-tuples* could be assumed to be true (with an optimistic approach) or could be assumed to be false (with a skeptical approach) – this leads to the null tuple. However, when such nonmonotonic inferences are made, the correctness of answers generated cannot be guaranteed.

4.2.1 Projection

The Projection operator (Π) is defined as follows:

$$a : \{C\} \, \epsilon \, \Pi \; (R, a_\pi) \longleftrightarrow in\_project(a : \{C\}, R, a_\pi)$$

where $a : \{C\}$ denotes an *mc-tuple* of the form $< a_1, .., a_n, C >$ and *in_project* is defined as follows:

$$in\_project(t : \{c_t\}, r, a_\pi) \leftarrow r \models x : \{c_x\} \wedge$$
$$(t = x[a_\pi]) \wedge (project\_constr(c_t, c_x, a_\pi))$$

Here *project_constr* is a predicate which is used to reflect the projection operation onto the constraint part of the *mc-tuples* in the relation.

4.3 Join

The behavior of the join operator in the presence of null values is slightly different from the behavior of the standard join operator. To see this, consider the `phone` relation with the schema $(Name, Phone)$ in Appendix A, and consider another relation, R_2 having the schema $(Office, Phone)$ meaning that $Phone$ is in the $Office$. Let us now see what happens when we attempt to join the following two *mc-tuples* from `phone` and $R2$, respectively.

> t_1 is in `phone` relation and t_2 is in R_2 where:
> $t_1 = < va\_mar(elaine), ex\_mar, true >$
> $t_2 = < va\_mar(1221), ma\_mar, true >$

It is not obvious how to do a join operation on the $Phone$ attribute because the semantics of the null markers residing in the *Phone*-slot are different. We now define an operation, called the *marker join operator*, that will facilitate the combination, via a join operation, of such *mc-tuples*. To see how this is implemented, we observe, in the above case, that the joined triple

$$< va\_mar(elaine), ?, 1221, true >$$

should exist in the resulting join relation iff the office room 1221 does indeed contain a phone in it, and that phone is the same phone that Elaine uses. (How to fill in the "?" above will become apparent as we proceed).

4.3.1 Marker Join Operator

If the joining attributes are $t_1.X$ and $t_2.Y$ where t_1 is an *mc-tuple* in relation R_1 and t_2 is an *mc-tuple* in relation R_2, then the marker join operator ($\odot_{mj}$) can be defined as follows (on the join attributes):

X	Y	J	Constraint
va_mar	va_mar	va_mar [1]	—
va_mar	ex_mar	va_mar	$(t_2.Y = a)$ [3]
va_mar	ma_mar	va_mar	$(t_2.Y = a)$ [3]
va_mar	pa_mar	va_mar [2]	$(t_2.Y = a)$ [3]
va_mar	pm_mar	va_mar [2]	$(t_2.Y = a)$ [3]
va_mar	pl_mar	—	—
ex_mar	ex_mar	ex_mar	$(t_1.X = t_2.Y = J)$
ex_mar	ma_mar	ex_mar	$(t_1.X = t_2.Y = J)$
ex_mar	pa_mar	pa_mar	$(t_1.X = t_2.Y = J)$
ex_mar	pm_mar	pa_mar	$(t_1.X = t_2.Y = J)$
ex_mar	pl_mar	—	—
ma_mar	ma_mar	ex_mar	$(t_1.X = t_2.Y = J)$
ma_mar	pa_mar	pa_mar	$(t_1.X = t_2.Y = J)$
ma_mar	pm_mar	pa_mar	$(t_1.X = t_2.Y = J)$
ma_mar	pl_mar	—	—
pa_mar	pa_mar	pa_mar	$(t_1.X = t_2.Y = J)$
pa_mar	pm_mar	pa_mar	$(t_1.X = t_2.Y = J)$
pa_mar	pl_mar	—	—
pl_mar	pl_mar	—	—
pl_mar	pm_mar	—	—
pm_mar	pm_mar	pa_mar	$(t_1.X = t_2.Y = J)$

1. If $va(X) = va(Y)$ then the *mc-tuple* obtained by performing the join is in the result, else it is not in the result.
2. If $va(X) \subset va(Y)$ then the *mc-tuple* obtained by performing the join is in the result, else it is not in the result.
3. a is the value stored in the attribute-slot X, i.e. $va(X) = \{a\}$.

The following example shows how the marker join operator is used in joining two tables:

Example 4.3 Let R_1 and R_2 be two null valued relations:

R_1:

tid	Person	Employer	Constraint
21	sherry	ncr	true
22	ed	$pa\_mar = \{ibm, ncr\}$	true
23	vic	ma_mar	true

R_2:

tid	Person-2	Employer	Constraint
71	kate	ibm	true
72	john	$pa\_mar = \{ibm, ncr\}$	true
73	kristina	ma_mar	true

Then the join of this two relations is:

	tid	Person	Employer	Person-2	Constraint
	91	sherry	ncr	john	$(72.Employer = ncr)$
	92	sherry	ncr	kristina	$(73.Employer = ncr)$
	93	ed	ibm	kate	$(71.Employer = ibm)$
$R_1 \bowtie R_2$:	94	ed	$pa\_mar = \{ibm, ncr\}$	john	$(22.Employer = 72.Employer = Employer)$
	95	ed	$pa\_mar = \{ibm, ncr\}$	kristina	$(22.Employer = 73.Employer = Employer)$
	96	vic	ibm	kate	$(23.Employer = ibm)$
	97	vic	$pa\_mar = \{ibm, ncr\}$	john	$(23.Employer = 72.Employer = Employer)$
	98	vic	ex_mar	kristina	$(23.Employer = 73.Employer = Employer)$

It is important to note that in some cases, the constraint part of the resulting *mc-tuple* t_3 contains references to one or both of the original *mc-tuples* participating in the join.

4.3.2 Joined Predicate

This is a ternary predicate of the form: $joined(t_1, t_2, t_3)$ where t_1, t_2 and t_3 are *mc-tuples*. The above atom is true iff t_3 is the result of applying the marker join operation on the joining attributes of *mc-tuples* t_1 and t_2, and taking the conjunction of the constraints of t_1 and t_2 and otherwise applying the regular join operation to the tuples.

4.3.3 Join

This operator takes two *mc-tuples* and applies the Marker Join Operator to all the joining attributes involved in them. The conjunction of the resulting "joining" constraints, as well as the original constraints in the two parent *mc-tuples* forms a new, composite constraint.

$$a \ \epsilon \bowtie (R_1, R_2) \longleftrightarrow in\_join(a, R_1, R_2)$$

where in_join is defined as follows:

$$\begin{aligned} in\_join(x, r_1, r_2) \leftarrow \ & r_1 \models y \wedge \\ & r_2 \models z \wedge joined(y, z, x) \end{aligned}$$

4.4 Union, Intersection and Difference

4.4.1 Union

The definition of the union operator is exactly as it was in the standard case.

$$a \ \epsilon \bigcup (R_1, R_2) \longleftrightarrow in\_union(a, R_1, R_2)$$

where in_union is defined as follows:

$$in\_union(x, r_1, r_2) \leftarrow (r_1 \models x) \vee (r_2 \models x)$$

4.4.2 Intersection

The definition of the intersection operation is somewhat different from the standard case. To see this, consider the case when $<va\_mar\ (20), c_1>$ is in R_1 and $<va\_mar\ (20), c_2>$ is in R_2. In this case we would expect the *mc-tuple* $<va\_mar\ (20), c_1 \wedge c_2>$ to be in the intersection. However, the standard definition of the intersection operation does not yield this result because it is not equipped to handle constraints. Hence, the definition of the intersection operator needs to be modified. We define it as follows:

$$a \in \bigcap (R_1, R_2, K) \longleftrightarrow in\_intersect(a, R_1, R_2, K)$$

where *in_intersect* is defined as follows:

$$\begin{aligned} in\_intersect(x : \{c_x\}, r_1, r_2, k) \leftarrow\ & (r_1 \models y : \{c_y\}) \wedge (r_2 \models z : \{c_z\}) \wedge \\ & (x = y \odot_i z)) \wedge \\ & (c_x = c_y \wedge c_z) \end{aligned}$$

The operator "$\odot_i$" is defined in such a way that for each attribute A,

$$va(x.A) = va(y.A) \bigcap va(z.A).$$

If this intersection results in an empty set for at least one of the attributes, then the *mc-tuple* is the null tuple.
Thus, in the example given above, the new definition yields $<va\_mar\ (20), c_1 \wedge c_2>$ in the intersection.

4.4.3 Difference

The Difference operator is also slightly different from its standard counterpart. It is defined as follows:

$$a \in Dif\ (R_1, R_2, K) \longleftrightarrow in\_dif(a, R_1, R_2, K)$$

where *in_dif* is defined as follows:

$$\begin{aligned} in\_dif(x : \{c_x\}, r_1, r_2, k) \leftarrow\ & (r_1 \models y : \{c_y\}) \wedge (r_2 \models z : \{c_z\}) \wedge \\ & (((x = y \odot_{d,A} z) \wedge\ (c_x = c_y)) \vee \\ & \ ((x = y \odot_{d,B} z) \wedge\ (c_x = c_y)) \vee \\ & \ \ldots \\ & \ ((x = y) \wedge\ (c_x = c_y \wedge \neg c_z))) \end{aligned}$$

For a specified attribute A, "$\odot_{d,A}$" is defined so as to satisfy the equality

$$va(x.A) = va(y.A) \bigcap ((Dom(A) \bigcup \{\perp\}) - va(z.A)).$$

If this intersection results in an empty set for the domain of the attribute A, then the associated *mc-tuple* is the null tuple.

Note that for every attribute in the schema, we have a conjunct in the difference operator.

An example will help the reader see how this operator works. Let $<va\_mar\ (20), c_1>$ be in R_1 and let $<va\_mar\ (20), c_2>$ be in R_2. In this case, the *mc-tuple* $<va\_mar\ (20), c_1 \wedge \neg c_2>$ will be in the difference.

Having described the basic, primitive operations in the null value algebra, we are now in a position to study the algebraic relationships that are true within this algebra. These algebraic relationships can be used for effective query optimization.

5 Properties of the Null-Valued Algebra

In the preceding sections, we have defined an algebra for databases containing null values. In this section, we establish various properties of this algebra. In the case of each such property, we discuss the impact of the property involved.

In the rest of this section, whenever op_1 and op_2 are two algebraic operators, $op_2 op_1$ denotes the application of op_1 followed by the subsequent application of op_2.

Property 1 *$Rep(\sigma_1\sigma_2) = Rep(\sigma_2\sigma_1)$.*

This property says that the order in which the selection operators are applied does not matter.

Proof : Basically, an *mc-tuple* t is in $\sigma_1\sigma_2$ if the following holds:

$$((R \models y) \wedge \Lambda(\Theta_2, y, x)) \wedge \Lambda(\Theta_1, x, t))$$

where R is the input relation.

Changing the order of the application of Θ_2 and Θ_1 will change the syntax of the resulting constraints, but the satisfiability of the two constraints will stay unchanged. Hence we have,

$$((R \models y) \wedge \Lambda(\Theta_2, y, x)) \wedge \Lambda(\Theta_2, x, t)) \longleftrightarrow ((R \models y) \wedge \Lambda(\Theta_1, y, x)) \wedge \Lambda(\Theta_2, x, t))$$

and the above equality holds.

□

Property 2 *$Rep(\Pi\sigma) = Rep(\sigma\Pi)$ (If the selection condition applies to the relation produced by the projection).*

Proof : Follows directly from the properties of the selection operator and the commutativity of the conjunction of constraints.

□

The above result says that the order in which projections and selections are done is not relevant (as long as the selection condition still applies to the result of the projection). An implication of this result is that whenever selections and projections are to be done one after the other, it may be better to do the projection operation first as we may then be able to eliminate various constraints that do not apply to the fields that we are projecting.

The next result is more interesting. It says, in effect, that converting relations (including interim relations) to canonical form commutes with projection.

Property 3 *Rep*($\mathcal{C}\Pi$) = *Rep*($\Pi\mathcal{C}$) *(if the union based semantics is used).*

Proof : Note that we assume that the compaction operators use the same quasi-keys.

When applied to a relation, the projection operator does not make any semantical changes to the constraints of *mc-tuples* , but only removes columns from the table. Hence, the behaviour of the compaction operator (which, in this case, only takes the union of the domains of the markers) is not affected by the projection operation. Hence, it is possible to perform the compaction operation before or after the projection without changing the result.

□

In the intersection based compaction semantics some tuples are omitted from the tuples after the compaction, due to their empty attribute domains. This fact makes it impossible to guarantee the above property.

In the standard relational model of data, it is well known that $\sigma(a \bowtie b) = \sigma(a) \bowtie \sigma(b)$. Similarly in our extended model the following holds:

Property 4 *Rep*($\sigma(a \bowtie b)$) = *Rep*($\sigma(a) \bowtie \sigma(b)$).

However, as shown by the following example, it may be hard to see this equality unless a database history is kept to keep track of the temporary relations, and the way they were created.

Example 5.1 *Let R_1 and R_2 be two relations with the following schemas:*

$R_1 < Name, Weight >$
$R_2 < Name, Weight >$

Let these relations contain the following mc-tuples :

$R_1 : t_1 =< va\_mar\ (John), ex\_mar\ , true >$
$R_2 : t_2 =< va\_mar\ (John), ex\_mar\ , true >$

Consider the selection condition $(Weight < 175)$*. Then:*

$$\begin{aligned} R_1 \bowtie R_2 &= < va\_mar\ (John), ex\_mar\ , (t_1.Weight = t_2.Weight = Weight) > . \\ \sigma(R_1 \bowtie R_2) &= < va\_mar\ (John), ex\_mar\ , (t_1.Weight = t_2.Weight = Weight) \wedge (Weight < 175) > . \end{aligned}$$

When computing $\sigma(R_1) \bowtie \sigma(R_2)$, we notice that:

$$\begin{aligned} \sigma(R_1) &= < va\_mar\ (John), ex\_mar\ , (t_1.Weight < 175) > \\ \sigma(R_2) &= < va\_mar\ (John), ex\_mar\ , (t_2.Weight < 175) > \\ \sigma(R_1) \bowtie \sigma(R_2) &= < va\_mar\ (John), ex\_mar\ , (t_1.Weight < 175) \wedge \\ &\qquad (t_2.Weight < 175) \wedge \\ &\qquad (t_{r1}.Weight = t_{r2}.Weight = Weight) > . \end{aligned}$$

□

Here t_{r1} and t_{r2} are *mc-tuples* in intermediary relations. It is easy to see that one of the results, viz. $\sigma(R_1) \bowtie \sigma(R_2)$ contains a reference to an intermediate tuple, while the other computation, $\sigma(R_1 \bowtie R_2)$does not contain such a reference.

Unfortunately, "join" need not be syntactically associative in the presence of multiple types of null values, though semantically it is:

Property 5 *Rep*$(a \bowtie (b \bowtie c)) =$ *Rep*$((a \bowtie b) \bowtie c)$.

To see when a syntactic difference may occur, consider the following example where R_1,R_2 and R_3 are relations with the following schemas:

$R_1 < Name, Height >$
$R_2 < SSN, Height >$
$R_3 < PIN, Height >$.

Example 5.2 *Let these relations contain the following mc-tuples :*

$R_1 : t_1 = <va\_mar\ (John), ex\_mar\ , true >$
$R_2 : t_2 = <va\_mar\ (211567842), ma\_mar\ , true >$
$R_3 : t_3 = <va\_mar\ (2578), va\_mar\ (6.0), true >$.

$$\begin{aligned}
R_2 \bowtie R_3 &= < va_mar\ (2578), va_mar\ (211567842), va_mar\ (6.0), (t_2.Height = 6.0) > \\
R_1 \bowtie (R_2 \bowtie R_3) &= < va_mar\ (John), va_mar\ (2578), va_mar\ (211567842), va_mar\ (6.0), \\
&\qquad (t_2.Height = 6.0) \wedge (t_1.Height = 6.0) > . \\
R_1 \bowtie R_2 &= < va_mar\ (John), va_mar\ (211567842), ex_mar\ , ((t_1.Height = t_2.Height = Height) > \\
(R_1 \bowtie R_2) \bowtie R_3 &= < va_mar\ (John), va_mar\ (2578), va_mar\ (211567842), va_mar\ (6.0), \\
&\qquad (t_1.Height = t_2.Height = Height) \wedge (t_r.Height = 6.0) > .
\end{aligned}$$

Here t_r is an *mc-tuple* in the intermediate relation. It is easy to see that the resulting *mc-tuples* are different.

The positive result here is that if a value attribute is involved in a join operation, and if this value marker is in the middle position (i.e. the join is of the form $X \bowtie(va\_mar \bowtie Y)$, then no temporary relation is needed.

Similarly the following equations hold.

Property 6 *Rep*$(\sigma(a \bigcup b)) =$ *Rep*$(\sigma(a) \bigcup \sigma(b))$.

Property 7 *Rep*$(\sigma(a \bigcap b)) =$ *Rep*$(\sigma(a) \bigcap \sigma(b))$.

Property 8 *Rep*$(\sigma(a\ Dif\ b)) =$ *Rep*$(\sigma(a)\ Dif\ \sigma(b))$.

Proof : The proofs of these properties are omitted.

□

6 Experimental Results on Query Optimization

We have developed an experimental implementation of a database containing null values of the types described in this paper. The implementation consists of a total of approximately 4000 lines of C-code and runs on a Sparc/Unix workstation. The implementation is not an implementation of a full-fledged DBMS; rather, it consists of a body of algorithms implementing the various selection, projection, join, and canonical form computation operations. Most of the code relates to managing the constraints that arise when null values are present. Below, we report on the result of four experiments we have conducted based on this prototype implementation.

All the experiments (except joins) use six relations $R_1, .., R_6$ having identical schemas

$$< A_1, A_2, A_3 >$$

where each of these attributes is of type integer. Relations (R_1) and (R_4) contain 250 tuples, relations (R_2) and (R_5) contain 1000 tuples and relations(R_3) and (R_6) contain 2000 tuples. The first attribute of each relation (A_1) contains only *va_mar* markers. This attribute is the quasi-key attribute. The other two attributes (A_2 and A_3) contain all possible markers (*va_mar* , *ex_mar* , *pm_mar* , *ma_mar* ,*pa_mar*) with equal probability (20% each) for R_1, R_2, R_3 and contain 50% *va_mar* , 50% (*ex_mar* , *pm_mar* , *ma_mar* ,*pa_mar*) for for R_4, R_5, R_6.

The experiments for joins are carried out on similar sets of relations. But this time there are two sets of relations with schemas $< A_1, A_2, A_3 >$ and $< A_1, A_2, A_4 >$ (so that, the join occurs on attributes A_1 and A_2) and the *va_mar* ratio is taken to be 33.3% and 50% in different experiments.

All times in this section are given in miliseconds.

6.1 Experiment 1

Purpose: The main aim of this experiment was to study the equality $\sigma_1\sigma_2 = \sigma_2\sigma_1$. In particular, we wished to determine whether it is better to first perform selections on relations with relatively few null values, or to perform them on relations with a larger number of null values.
Method: We used two sets of timings. In the first, we let σ_1 be ($A_1 = A_2$), and σ_2 be ($A_2 = A_3$). Thus, both the cascaded selections, $\sigma_1\sigma_2$ and $\sigma_2\sigma_1$ have the same selection condition, viz. ($A_1 = A_2 = A_3$). The result of these experiments is shown in Figure 1.

In the second set of timings, the only change we made was that σ_2 selected all tuples where $A_1 = A_3$. The net result of the cascaded selects, $\sigma_1\sigma_2$ and $\sigma_2\sigma_1$ is still the set of all tuples satisfying the condition ($A_1 = A_2 = A_3$). The result of these timings is shown in Figure 2.
Interpretation of Results: In the first set of timings (cf. Figure 1), the cascaded selection $\sigma_1\sigma_2$ first performs selection on the condition $A_2 = A_3$. This selection condition operates on attibute columns containing a relatively large number of null values (and hence constraints). Consequently, it is relatively hard to eliminate tuples. In contrast, the cascaded selection $\sigma_2\sigma_1$ first applies the selection condition σ_1 which checks if $A_1 = A_2$. The attribute field A_1 contains only *va_mar* markers, i.e. it contains no null values. Thus, a relatively large number of tuples can be eliminated by this selection condition *prior* to performing a selection (viz. σ_2) that operates on attribute fields containing relatively large numbers of null values. Thus, the times recorded for the cascaded selection $\sigma_2\sigma_1$ are better than for the cascaded selection $\sigma_1\sigma_2$.

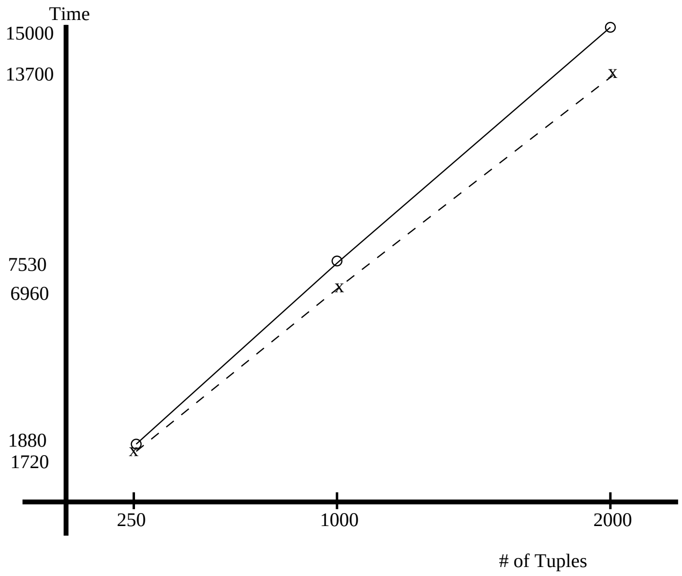


Figure 1: $\sigma_2\sigma_1$ is shown by dotted lines (- - -), $\sigma_1\sigma_2$ is shown by solid lines (—)

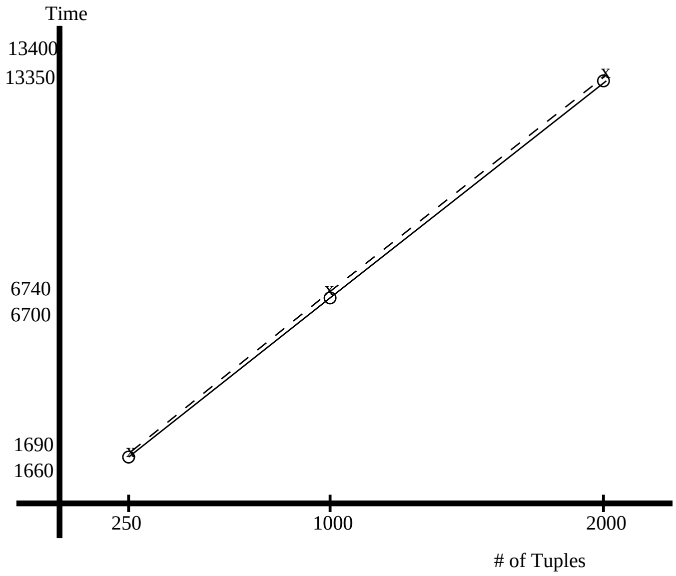


Figure 2: $\sigma_2\sigma_1$ is shown by dotted lines (- - -), $\sigma_1\sigma_2$ is shown by solid lines (—)

In contrast, consider Figure 2, reflecting the second set of timing data. Here, *both* σ_1 and σ_2 make comparisons against the attribute A_1 which contains only value markers. Thus, both the selection conditions $A_1 = A_2$ and $A_1 = A_3$ eliminate relatively even numbers of tuples, and hence, the cascaded selections. $\sigma_1\sigma_2$ and $\sigma_2\sigma_1$ yield almost identical results.

The observant reader will notice that all times in Figure 2 are lower than in Figure 1. This is not an accident – rewriting cascaded selects so as to make all comparisons apply to at least one field that has only (or mostly) value markers leads to a significant savings in time.
Impact on Query Optimization: Whenever a selection of the form $A_1 = A_2 = A_3 = \ldots = A_n$ is being performed, if one of the attributes, say A_i, consists entirely (or almost entirely) of value markers, then this set of equalities should be computed as the cascaded select

$$\sigma_1^* \ldots \sigma_{i-1}^* \sigma_{i+1}^* \ldots \sigma_n^*$$

where σ_j^* uses the selection condition $(A_i = A_j)$.

6.2 Experiment 2

Purpose: The main aim of this experiment was to determine how the overall performance changed when the attributes A_2 and A_3 had half of their slots containing value markers, with the other half being evenly distributed among the other types of markers.
Method: The same two sets of timings as in Experiment 1 were taken.
Interpretation of Results: Figures 3 and 4 show the results using the same two sets of timings as in Experiment 1. As the reader will observe, the observations of Experiment 1 continue to hold here. Furthermore, when the number of value markers is increased (from 20% in Experiment 1 to 50% in Experiment 2), the overall processing time for the cascaded selections drops. This is because the presence of value markers causes a large number of tuples to be eliminated, thus eliminating the need to manage various associated constraints.

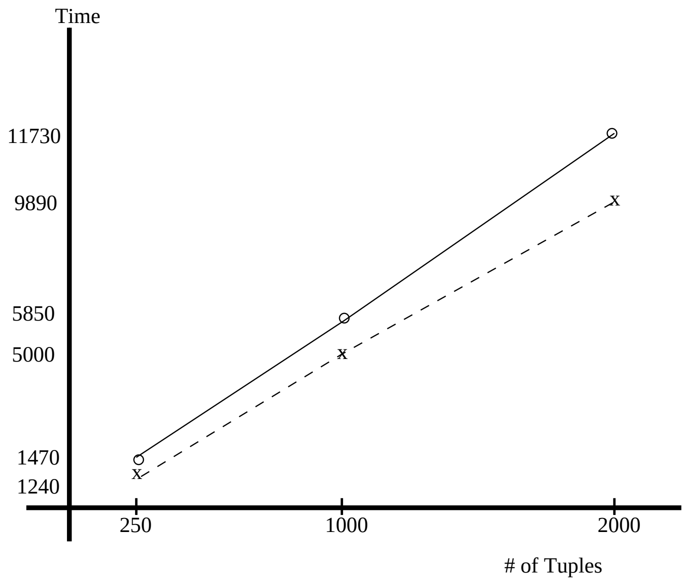


Figure 3: $\sigma_2\sigma_1$ is shown by dotted lines (- - -), $\sigma_1\sigma_2$ is shown by solid lines (—)

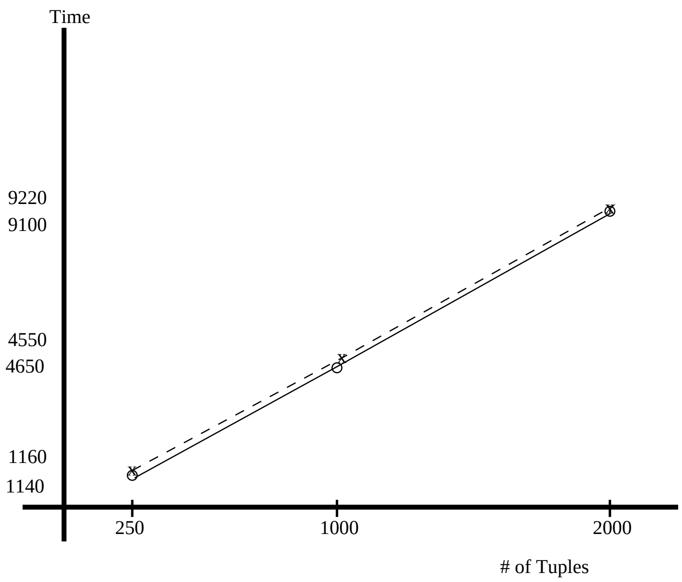


Figure 4: $\sigma_2\sigma_1$ is shown by dotted lines (- - -), $\sigma_1\sigma_2$ is shown by solid lines (—)

Impact on Query Optimization: The performance of the system increases as the number of value markers involved in the selection criterion increase. Thus, it is better to do selections first on attributes that have a large proportion of value markers.

6.3 Experiment 3

Purpose: The purpose of the third experiment was to study the equality $\Pi\sigma = \sigma\Pi$, and to determine which order was better when computing these expressions in databases that contain null values.
Method: In this case, we took two sets of timing data. First, we took σ to be $A_1 = A_2$ and Π to project on the attributes A_1 and A_2. Hence, the result of the projection does not affect the selection condition.

Second, we increased (as in Experiment 2) the proportion of value markers in A_2 and A_3 to 50%, with the other null values being evenly distributed over the remaining slots.
Interpretation of Results: Figure 5 contains the results of this experiment. It is clear from this figure that performing the projection operation first decreases the overall average execution time.

The second set of timing data is shown in Figure 6. Note that the total times taken are smaller than in the case of the first timing data.
Impact on Query Optimization: When we consider an expression of the form $\Pi\sigma$, then we are better off rewriting this expression as $\sigma\Pi$ when possible. As the number of value markers increase, the impact of this rewriting is likely to become more and more significant.

6.4 Experiment 4

Purpose: The aim of this experiment was to study the equality $\sigma(a \bowtie b) = \sigma(a) \bowtie \sigma(b)$. Recall that these two operations are equal when the attribute b contains value markers only (otherwise these two expressions are not equal).

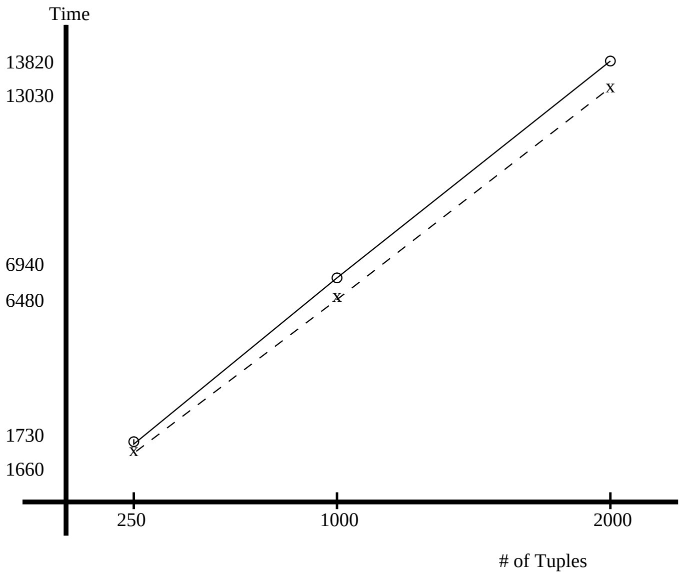


Figure 5: $\sigma\Pi$ is shown by dotted lines (- - -), $\Pi\sigma$ is shown by solid lines (—)

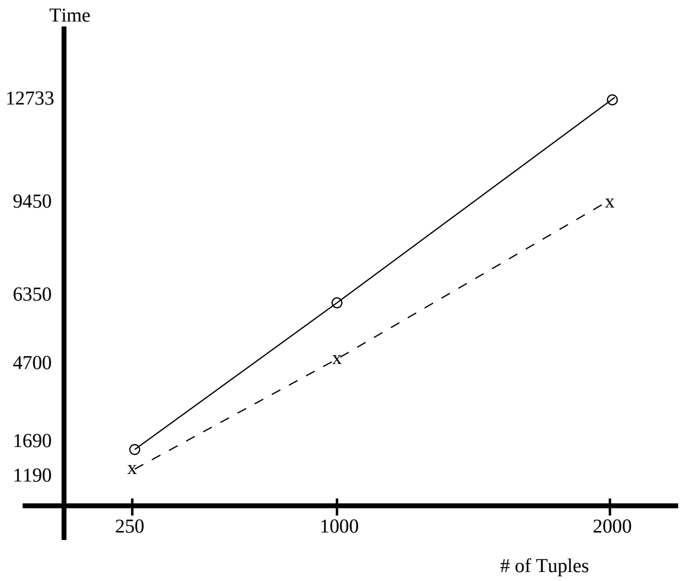


Figure 6: $\sigma(a \bowtie b)$ is shown by dotted lines (- - -), $\sigma(a) \bowtie \sigma(b)$ is shown by solid lines (—)

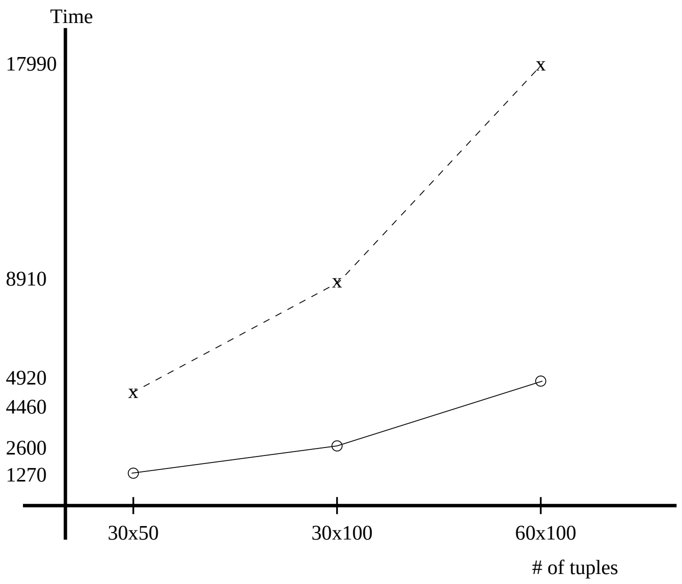


Figure 7: $\sigma(a \bowtie b)$ is shown by dotted lines (- - -), $\sigma(a) \bowtie \sigma(b)$ is shown by solid lines (—)

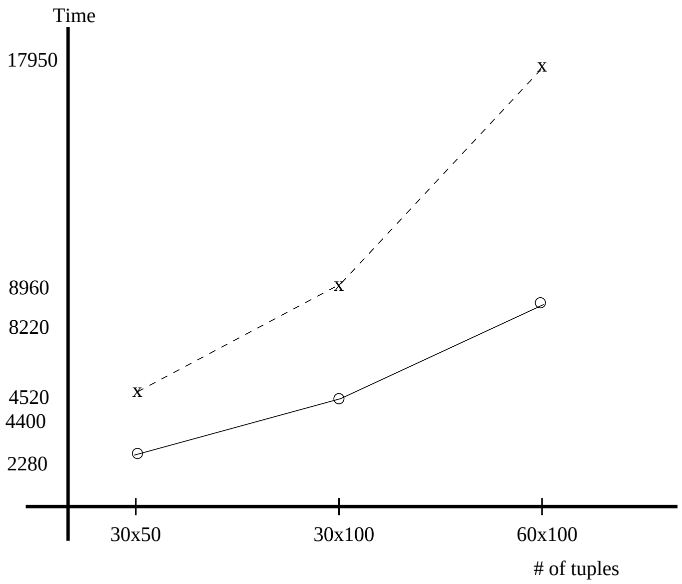


Figure 8: $\sigma(a \bowtie b)$ is shown by dotted lines (- - -), $\sigma(a) \bowtie \sigma(b)$ is shown by solid lines (—)

Method: The experiments have been performed over two sets of randomly generated relations as described in the beginning of Section 6.
Interpretation of Results: The results are contained in Figure 7 and 8. They show that when the relations contain a relatively high proportion of value markers, then performing the selections before performing the join is advantageous. The reason for this is that the selections may eliminate many tuples, thus performing the join operate on two relatively small relations. Furthermore, as the number of value markers involved in the attribute on which the selections are performed increases, the advantage of performing selections first, before doing the join becomes more and more pronounced.
Impact on Query Optimization: The results imply that we are almost always better off doing selections before doing joins.

7 Related Work

In this section we provide a survey of work on null values related to this paper. We exclude papers that deal primarily with complexity issues or other topics such as the universal relation concept. Table 1 shows the types of null values considered by other authors and serves as a starting point for our survey. We also include entries for other types of nulls and for papers that deal with constraints and correctness. We are not aware of prior research on null values that involved experimentation for query processing with different types of nulls.

Author-Year	Ex.	Maybe	Plc.-H.	Partial	Part. M'be	Add.	Constr.	Corr.	Expts
ANSI/X3/SPARC 75	√		√			Operational information(available, derived)			
Codd 75,79	√								
Grant 77	√		√						
Grant 79,80	√			√					
Lipski 79,81	√			√					
Imielinski-Lipski 81,84	√						√	√	
Lien 79			√						
Vassiliou 79	√		√						
Zaniolo 84	√	√	√						
Wong 82						Probability distribution			
Biskup 83,84	√					Universal null, Maybe tuples		√	
Reiter 84,86	√							√	
Yuang-Chiang 87	√					Disjunctive information		√	
Grant-Minker 86	√					Disjunctive information			
Minker-Perlis 85						Protected data		√	
Liu-Sunderraman 97,90,91						Disjunctive information, Protected data		√	
Atzeni-DeAntonellis 93	√	√	√	√	√	Probability distribution	√		
Abiteboul-hull-Vianu 95	√						√	√	
Candan-Grant-Subrahmanian	√	√	√	√	√		√	√	√

Research in null values began with the ANSI/X3/SPARC report [2] that distinguished among 14 types of nulls. However most of these types are special cases of our existential and place holder nulls, the others are operational definitions such as "available, but of suspect validity(unreliable)". The first paper that deals with the handling of null values in query processing is Codd[7]. A 3-valued logic is introduced for the handling of existential nulls. Grant [9] points out a flaw in this method and suggests a method to solve this problem as well as to deal with placeholder nulls. Additional early work on partial nulls appears in Grant[10, 11], Lipski [16, 17].

Conditions (constraints) were introduced into tables in Imielinski-Lipski[13, 14] in connection with representation systems. Variables represent null values in these tables; constraints involving equalities and inequalities of variables and constraints may be associated with individual rows and the whole table. The constraints are used to limit the allowed interpretations for existential nulls. A table T with null values is assumed to stand for all tables that would be obtained from the table by substituting constant values for the variables in accordance with the constraints. The set of tables (without nulls) that T represents is written as $rep(T)$. For a query q that may involve a certain set of operations of the relational algebra, the set of answers to q on T may be represented by $q(rep(T))$. A table $q'(T)$ correctly represents the query if $rep(q'(T)) = q(rep(T))$. It is shown that queries in the relational algebra can be represented correctly by the set of conditional tables.

Building on the work of Lien[15] and Vassiliou [24], Zaniolo [27] introduces the maybe null, as the "no information" null. The operations of the relational algebra are generalized to this framework. Another approach to the null value problem is formulated in Wong[25] who assumes a probability distribution for an unknown value in a domain. Biskup [6] introduces the universal "don't care" null in analogy to the existential null. in both Biskup[5] and [6], the correctness of the operations are proved and "maybe tuples" from previous operations are allowed and used in a systematic way.

Reiter[22] proposed a formal theory of databases in first-order logic including existential nulls. Existential nulls are treated as Skolem constants without unique name axioms. Within this framework a correct, but incomplete query evaluation algorithm is given for the relational calculus by Reiter[23]. Yuan-Chiang[26] extend the work of Reiter; their algorithm is complete and allows indefinite information in the form of a disjunction. Grant-Minker[12] also provide an algorithm for finding the answers to a query in a disjunctive database with negation interpreted through the Generalized Closed World Assumption. In some cases, the Closed World Assumption and its variants allow too much negative information to be deduced, hence the notion of protection for atoms was introduced, that is, protection from assuming the negation of the atom. A query evaluation algorithm is given for such situations by Minker and Perlis[21].

Among recent papers the ones by Liu-Sunderraman [18, 19, 20] deal with indefinite information in the sense of disjunctive tuples as well as protection in the sense of maybe tuples. The relational algebra is generalized to what are called I-tables, and correctness is proved. There are also several recent books on relational database theory that contain information on null values. Atzeni-DeAntonellis [3, Chapter 6] deals with all five types of nulls. The representation uses first-order formulas with a regular existential quantifier; in some cases additional predicates are needed omitting the attribute on which the null values, like the placeholder null, occurs. Implication formulas must also be set up among the predicates. Probability distribution on the domain and constraints are also introduced for existential nulls. Abiteboul-Hull-Vianu [1, Chapter 19] deal mostly with the Imielinski-Lipski work and also include material and references on complexity issues.

8 Conclusions

A frequent occurrence in relational databases is that certain attribute slots in tuples cannot be filled in for any of a number of reasons. These reasons could include the fact that a value exists but is not known (existential null), a value may or may not exist (maybe null), it is known that a value does not exists and this field is inapplicable to the tuple being considered (placeholder null), a value exists and is known to be within a specified set (partial null), and it is not known whether a value exists or not, but if it does, it must fall within a specified set (partial placeholder null). Despite the fact that the existence of these different null values has been noted for a long time (at least twenty years), no unified treatment of these different null values has emerged.

An important start in this direction was made by Imielinski and Lipski who developed a notion of *condition tables* where tuples had associated conditions. The tuples were "in" the given relation only if the affiliated condition was true. Imielinski and Lipski used these intuitions to develop an elegant treatment of one kind of null value, viz. the existential null. *In this paper, we have shown how constraints may be used to provide a unified treatment of all the types of nulls considered above.* Though most of these null values have been treated individually (e.g. [3]), these treatments have considered the respective null values in isolation, and have not provided a single unifying framework.

Based on our unified constraint-based model, we have developed an algebra for databases containing these varied types of null values. We have studied various mathematical aspects of this algebra, and have, in particular, established various equivalences. We have developed a prototype implementation of these different types of null values, and used this implementation as an experimental testbed to evaluate alternative query evaluation strategies when null values are present.

References

[1] Abiteboul, S., Hull, R., Vianu, V., Foundations of Databases, Addison-Wesley Publishing Company, 1995.

[2] ANSI/X3/SPARC Study Group on Data Base Management Systems, Interim Report 75-02-08, FDT Bulletin of ACM-SIGMOD Vol. 7, No. 2, 1975.

[3] Atzeni, P. and De Antonellis, V., Relational Database Theory, The Benjamin/Cummings Publishing Company, Inc., 1993.

[4] Biskup, J., A Formal Approach to Null Values in Database Relations, In: Advances in Database Theory, Volume 1 (H. Gallaire, J. Minker, and J. M. Nicolas, Eds.) Plenum Press, 1981, pp 299-341.

[5] Biskup, J., A Foundation of Codd's Relational Maybe-Operations, ACM TODS 8 (1983) pp. 608-636.

[6] Biskup, J., Extending the Relational Algebra for Relations with Maybe Tuples and Existential and Universal Null Values, Fundamenta Informaticae VII (1984) pp. 129-150.

[7] Codd, E. F., Understanding Relations (Installment #7) FDT Bulletin of ACM-SIGMOD 7 (1975) pp. 23-28.

[8] Codd, E. F., Extending the Database Relational Model to Capture More Meaning, ACM TODS 4 (1979) pp. 397-434.

[9] Grant, J., Null Values in a Relational Data Base, Information Processing Letters 6 (1977) pp. 156-157.

[10] Grant, J., Partial Values in a Tabular Database Model, Information Processing Letters 9 (1979) pp. 97-99.

[11] Grant, J., Incomplete Information in a Relational Database, Fundamenta Informaticae III (1980) pp. 363-378.

[12] Grant, J. and Minker, J., Answering Queries in Indefinite Databases and the Null Value Problem, In: Advances in Computing Research Volume 3 (P. Kanellakis, Ed.) 1986 pp. 247-267.

[13] Imielinski, T. and Lipski, W., On Representing Incomplete Information in a Relational Database, Proc. of 7th VLDB Conf., 1981, pp. 389-397.

[14] Imielinski, T. and Lipski, W., Incomplete Information in Relational Databases, JACM 31 (1984) pp. 761-791.

[15] Lien, E., Multivalued Dependencies with Null Values in Relational Databases, Proc. of 5th VLDB Conf., 1979, pp. 61-66.

[16] Lipski, W., On Semantic Issues Connected with Incomplete Information Systems, ACM TODS 4 (1979) pp. 262-296.

[17] Lipski, W., On Databases with Incomplete Information, JACM 28 (1981) pp. 41-70.

[18] Liu, K. C. and Sunderraman, R., An Extension to the Relational Model for Indefinite Databases, Proc. of ACM-IEEE Computer Society Joint Computer Conference 1987, pp. 428-435.

[19] Liu, K. C. and Sunderraman, R., Indefinite and Maybe Information in Relational Databases, ACM TODS 15 (1990) pp. 1-39.

[20] Liu, K. C. and Sunderraman, R., A Generalized Relational Model for Indefinite and Maybe Information, IEEE TKDE 3 (1991) pp. 65-77.

[21] Minker, J. and Perlis, D., Computing Protected Circumscription, Journal of Logic Programming 4 (1985) pp. 235-249.

[22] Reiter, R., Towards a Logical Reconstruction of Relational Database Theory, In: Conceptual Modeling (M. Brodie, J. Mylopoulos, and J. W. Schmidt, Eds.) Springer-Verlag, 1984, pp. 191-238.

[23] Reiter, R., A Sound and Sometimes Complete Query Evaluation Algorithm for Relational Databases with Null Values, JACM 33 (1986) pp. 349-370.

[24] Vassiliou, Y., Null Values in Database Management, A Denotational Semantics Approach, Proc. of ACM-SIGMOD 1979, pp. 162-169.

[25] E. Wong. A Statistical Approach to Incomplete Information in Database Systems, ACM TODS 7 (1982) pp. 470-488.

[26] Yuan, L. Y. and Chiang, D.-A., A Sound and Complete Query Evaluation Algorithm for Relational Databases with Null Values, Proc. 1989 ACM Symp. on Principles of Database Systems, 1989, pps 66–74.

[27] Zaniolo, C., Database Relations with Null Values, Journal of Computer and System Sciences 29 (1984), pp. 142-166.

Appendix A: A Sample Database

Relation `phone`:

tid	Name	Phone	Constraint
1	john	927 5872	true
2	tony	996 1873	true
3	ed	ma_mar	true
4	lisa	926 2890	true
5	elaine	ex_mar	true
6	irene	789 1892	true
7	david	$pa\_mar = \{926\ 2890,\ 593\ 1340\}$	true

Relation `spouse`:

tid	Husband	Wife	Constraint
11	john	sherry	true
12	ed	ex_mar	$(Wife = alice \vee Wife = susan)$
13	tony	pl_mar	true
14	vic	$pa\_mar = \{lisa, joan\}$	true
15	david	ma_mar	true
16	oscar	elaine	true

Relation `emp`:

tid	Person	Employer	Constraint
21	john	ibm	true
22	sherry	ncr	true
23	ed	$pa\_mar = \{ibm, ncr\}$	true
24	irene	ibm	true
25	vic	ma_mar	true
26	oscar	$pm\_mar = \{ncr\}$	true

Appendix B: Sample Database after some algebraic operations

Relation $\texttt{phone}_2$ after selection with condition $Phone = 927\ 5872$:

tid	Name	Phone	Constraint
$1^\bullet$	john	927 5872	true
$2^\bullet$	ed	ma_mar	$(Phone = 927\ 5872)$
$3^\bullet$	elaine	ex_mar	$(Phone = 927\ 5872)$

Relation $\texttt{spouse}_2$ (relation `spouse` after projection on attribute "Husband").

tid	Husband	Constraint
$11^\bullet$	john	true
$12^\bullet$	ed	$(\texttt{spouse}.12.Wife = alice \vee \texttt{spouse}.12.Wife = susan)$
$13^\bullet$	tony	true
$14^\bullet$	vic	true
$15^\bullet$	david	true
$16^\bullet$	oscar	true

Relation $\texttt{emp}_2$ after selection with condition $Employer = ncr$:

tid	Person	Employer	Constraint
$22^\bullet$	sherry	ncr	true
$23^\bullet$	ed	$pa\_mar = \{ibm, ncr\}$	$(Employer = ncr)$
$25^\bullet$	vic	ma_mar	$(Employer = ncr)$
$26^\bullet$	oscar	$pm\_mar = \{ncr\}$	$(Employer = ncr)$