Null Values in Relational Databases and Sure Information Answers

Hans-Joachim Klein

Institut für Informatik und Prakt. Mathematik
Universität Kiel, 24098 Kiel, Germany

Abstract. Null values are ubiquitous in database applications. There is, however, no common agreement upon how to deal with null values, neither in practice nor in theory. In this paper the null value problem is revisited with special emphasis on answers representing sure information with respect to possible world semantics. The focus is on the *unknown* and the *no information* interpretation of null values. A new semantics for the *no information* interpretation of nulls is proposed taking into account a problem arising in connection with the closed world assumption. Results are applied to the query language SQL.

1 Introduction

In one of his early papers on the relational data model, E.F. Codd stated: "The usual distinction between the primary key and other candidate keys (if any) is that no tuple is allowed to have an undefined value for any of the primary key components, whereas any other components may have an undefined value."([Cod72a], p. 37). In a subsequent paper he discussed relational algebra and tuple relational calculus and showed their equivalence without mentioning undefined values ([Cod72b]). This state of "secondary importance" of incompleteness caused by missing data values in attributes persisted in the forthcoming discussion of the classical relational data model.

The record oriented processing of data by "legacy" database systems puts responsibility for the handling of missing values to the application programmer. The close connection between CODASYL-systems and the programming language COBOL, for example, allows to use the figurative constant *low-value* of COBOL for representing and processing missing values. There are strong advocates for adopting a similar solution in the relational data model by providing *defaults* (special values) in domains ([DD98]). We do not consider this approach but concentrate on facilities to support the handling of missing information by the database system.

The main difference between concrete values of a domain and undefined values is the metalevel character of the latter. Since values in a database represent observations made in the "real world", undefined values correspond to certain phenomena of "reality". Examples are the missing knowledge about the birthday of a person, the non-existence of a maiden name for an unmarried person, or the missing of an expected news. The reason why such a phenomenon is not represented in the database by an ordinary data value can be well–known or unknown. Among the well–known causes is the mismatch between the granularity of facts and their representation in a database. This mismatch shows up, for example, as anomalies in connection with transitive functional dependencies or it is a consequence of special views such as the universal relation view ([MUV84]). In these cases, a value may be undefined since there is no corresponding phenomenon in "reality". This means that the knowledge about the value is complete (its non–existence coincides with an observed fact) but not representable by an ordinary value due to the chosen information structure. Null values marking such *inapplicable properties* ([Cod86]) are also called *non–applicable* or *nothing nulls* ([LeLi86], [Vas79]).

The uniform interpretation of missing values as *value existent but unknown* has been assumed for most of the approaches dealing with incomplete information in relational databases. Usually, no distinction is made between permanently and temporarily unknown values (for an exception see [GM90]). In [Cod79] a duplicate removal rule and the use of a three–valued logic were proposed obviously with influence on the null value concept of SQL ([ISO92]). Critique on these proposals originated in different perspectives such as the incompleteness of query evaluation ([Grt77]) and semantical problems in connection with the duplicate removal rule ([Bis83]).

A thorough investigation of representation problems in connection with the *unknown* interpretation of missing values and the evaluation of queries can be found in [IL84]. From there we know that in general one must be satisfied with approximations for the representation of answers and that working with variables (marked nulls) instead of a single null value often results in better approximations. The necessity to approximate answers even if there are no problems with their representation follows from complexity problems connected with incompleteness ([Var86]). Several proposals can be found in the literature to circumvent these problems with the consequence that query results are either incomplete ([Cod79], [Rei86], [Var86]) or may contain uncertain information ([Lak89]) not to mention erroneous proposals (see [ZP97] for comments into this regard). Another approach is to focus on a more basic kind of null value by assuming that there is no information at all if some value is missing ([Zan84]). Efficient evaluation strategies have been proposed for this interpretation.

In order to deal with different kinds of null values some approaches have focused on the introduction of special logics and operators (e.g.: [GZ88], [Ges90], [Yue91]). A deep understanding of the consequences of their application is necessary in order to be able to interpret query results correctly. This may be a major obstacle to their use.

A different level of incompleteness in relational databases is the missing of tuples in relations. Because the number of negative facts in a domain of application is usually very large it is common practice to store only positive facts and to evaluate queries under the so–called *closed world assumption* ([Rei78]). If this assumption shall be weakened in order to take some incomplete knowledge about the validity of facts into account, relations can be partitioned appropriately. Proposals into this direction can be found in [LS90] and [OO93], for example. We refer to such a partioning only in connection with query evaluation but assume the structure of relations in database to be homogeneous.

There are a number of interesting contributions in the field of incomplete relational databases we do not cover here. Overviews can be found in recent textbooks on database systems (e.g.: [LeLo99]). A source presenting a broad survey in the area of imperfect information is [MS97].

In this paper, our focus is on sure information answers in case of missing values interpreted as *value existent but unknown* or *there is no information*. In the next section we first discuss some basic concepts in connection with the modeling of incomplete information by a single special value and introduce some notation. Then we investigate the representation problem for answers to queries in a more general form and consider some approximations to sure answers based on model–theoretic semantics of incomplete databases. Section 4 deals with existential nulls. Complexity and decidability problems are pointed out and efficient but incomplete evaluation methods are discussed. Some research in this field is revisited under the viewpoint of sure information answers. No information nulls and null values in SQL are investigated in Section 5. It is shown that the correct handling of no information nulls with respect to a model–theoretic semantics has to solve problems implied by the closed world assumption. We make a proposal how to cope with this problem and point out some peculiarities of SQL semantics in connection with null values. Then a method for the transformation of SQL queries into query formulations approximating sure information answers is sketched. We conclude with some remarks.

2 Modeling Incomplete Information

2.1 Domain Extensions

A major advantage of the classical relational data model is its simplicity. Information is structured in finite sets of finite tuples belonging to the same type. Tuples are composed of atomic values taken from specified domains. A domain is a simple not necessarily finite set possibly with an order relation on it. Its elements are treated as uninterpreted objects.

If the model has to be generalized in order to cope with incomplete information not representable in this framework and if such a generalization shall preserve the main characteristics of the model, the domain concept can be enriched. Special domains with values specifying the information content of tuples can be introduced or existing domains can be extended by distinguished elements.

Additional attributes with special domains for the specification of the information content of tuples have been considered in [Bis83] and [IL84]. Entries into these attributes are generated during query evaluation to represent knowledge about the inclusion of tuples in results. The additional attributes, however, are not used in query formulations.

Our focus is on the extension of domains by a single distinguished value with the interpretation *value existent but unknown* (*unknown*, for short) or *there is no information on the value*. We consider both interpretations separately, i.e. we do not mix occurrences of different kinds of null values in databases. The *unknown* interpretation has been assumed in most of the studies dealing with incomplete information in relational databases and fits well to many applications. The *no information* interpretation may be considered as more basic than the *unknown* and the *non–applicable* interpretation ([Zan84]). It assumes that a missing value represents the information that no corresponding value exists in "reality" or that one or more values exist which are not known. This assumption is closely related to the null value concept of SQL.

Another important approach which should be mentioned is the extension of every domain by an infinite set of *variables* (Skolem constants) which allows to model information on the equality of unknown values ([IL84]). We sometimes refer to this kind of extension but do not consider it in detail.

2.2 Notation and Basics

We review first some notations for relational databases used in the sequel.
For our investigations it is sufficient to assume a single countably infinite domain of values D to be given.

Let a finite set $\alpha = \{A_1, ..., A_k\}, k \geq 1$, of *attributes* be given. A *relational database schema* σ on α is a finite set $\{(id_1, \alpha_1), ..., (id_m, \alpha_m)\}, m \geq 1$, of *relation types* $(id_i, \alpha_i), i \in \{1, ..., m\}$ where id_i is a unique identifier and α_i a subset of α. $(id_i, \alpha_i), \alpha_i = \{A_{i_1}, ..., A_{i_l}\}$, is also denoted as $id_i(A_{i_1}, ..., A_{i_l})$.
A *relation* on a subset α_i of α is a finite set of mappings $t \mid \alpha_i \rightarrow D^l, l = \mid \alpha_i \mid$. Its elements are called *tuples* over α_i. A *database state* or *database* over some database schema $\sigma = \{(id_1, \alpha_1), ..., (id_m, \alpha_m)\}$ is a set of relations $\{R_1, ..., R_m\}$, with one relation R_i on α_i for every relation type $(id_i, \alpha_i), i = 1, ..., m$.
A relation is called *incomplete* if some of its tuples are partial mappings; otherwise it is called *complete*. A database is called *incomplete*, if some of its relations are incomplete; otherwise, it is called *complete*. We use *idb* to denote incomplete databases and *pdb* for complete databases.

To simplify investigations it is assumed that different occurrences of missing values in databases are unrelated.
Independent of the interpretation of partiality a so–called information order can be introduced on sets of tuples over the same set of attributes. Let t, t' be two partial tuples over α_i. t' *subsumes* t (or is *more informative* than t), written $t' \geq t$, $\Leftrightarrow_{def}$ $t(A)$ defined $\Rightarrow t'(A) = t(A)$ for every $A \in \alpha_i$. If t' is a total tuple then it is called an *extension* of t.
In operations it is sometimes necessary to check for tuples having identical subsumptions. We therefore define: Let t', t be as above; t' and t are *compatible*, written $t' \Delta t$, if there is some tuple t'' over α_i such that $t'' \geq t'$ and $t'' \geq t$.

The information order on tuples can be extended to relations in several ways reflecting different semantics of incomplete relations. We distinguish the following alternatives: closed world (CWA) or open world assumption (OWA), a missing value is of type *no information (ni)* or *unknown (unk)*, and an unknown value represents exactly one ($\exists_1$) or at least one ($\exists_{\geq 1}$) concrete value. From the eight possible cases $((OWA, ni, \exists_1)$ and $(OWA, unk, \exists_1))$ can be discarded since they are covered by $(OWA, ni, \exists_{\geq 1})$ and $(OWA, unk, \exists_{\geq 1})$, resp. This leaves us with the following alternatives (up to close subsumption see [Mai83]):

Let R, S be two incomplete relations on the same set of attributes.

$(OWA, ni, \exists_{\geq 1})$: R is a *subsumption* of S if the following holds: $(\forall t \in S)(\exists t' \in R)(t' \geq t)$

$(OWA, unk, \exists_{\geq 1})$: R is an *extension* of S if R is a subsumption of S and total.

$(CWA, ni, \exists_{\geq 1})$: R is a *close subsumption* of S if R can be obtained from S by replacing some occurrences of undefined values by one or more concrete values in each case.

$(CWA, unk, \exists_{\geq 1})$: R is a *close extension* of S if R is an extension of S and the following holds: $(\forall t' \in R)(\exists t \in S)(t' \geq t)$

$(CWA, ni, \exists_1)$: R is an *augmentation* of S if R can be obtained from S by replacing some occurrences of undefined values by a concrete value in each case.

$(CWA, unk, \exists_1)$: R is a *completion* of S if R is an augmentation of S and total.

Example 1: For the incomplete relations P and Q below with attributes ranging over the natural numbers, two completions (R_1, R_2), an augmentation (R_3), two close extensions (R_4, R_5), and a close subsumption (R_6) are shown for P and Q, resp. ω denotes undefined values in P and Q; its semantics is *value not existent* in R_3 and R_6.

P

A	B	C
1	1	ω
1	2	ω
1	3	3

R_1

A	B	C
1	1	2
1	2	1
1	3	3

R_2

A	B	C
1	1	4
1	2	4
1	3	3

R_3

A	B	C
1	1	4
1	2	ω
1	3	3

Q

A	B	C
ω	1	3
ω	1	ω
2	1	ω

R_4

A	B	C
1	1	3
3	1	4
2	1	1
2	1	3

R_5

A	B	C
2	1	3
2	1	4

R_6

A	B	C
1	1	3
ω	1	2
ω	1	3
2	1	ω

We are interested in CWA–semantics which can now be defined for an incomplete relation R as the set of close subsumptions, close extensions, augmentations, or completions depending on the interpretation of incompleteness. Under the *no information* interpretation null values can be replaced either by one (non–key attribute) or several (key–attribute) concrete values or the null value remains unchanged and is interpreted as *value not existent*. If null values are allowed in key–attributes of an incomplete relation R then semantics is given by the set of close subsumptions of R; otherwise, semantics of R is given by the set of augmentations of R. For the *unknown* interpretation we have to take the set of close extensions and the set of completions of R, resp. Note that close subsumptions and augmentations represent complete information although they may be considered as incomplete relations at the syntactical level. We use the notations $POSS(R)$ and $POSS(idb)$ if the concrete kind of semantics is not important. The elements of $POSS(idb)$ are also called *possible worlds* to idb.

3 Query Semantics

3.1 Sure and Maybe Information in Answers

The discussion of query semantics based upon possible world semantics for incomplete databases can first be done without referring to concrete semantics and query languages. It is sufficient to consider queries as total functions mapping databases to relations and to assume *non–existent* values to be represented by a unique symbol of the domain D. For the following we suppose the reader to be familiar with relational algebra and relational calculus.

In case of complete databases, a query formulated in one of the common query languages against some schema σ can be considered as a mapping $q : S_\sigma \rightarrow \mathcal{R}_\beta$, where S_σ is the set of databases (or states) for σ and $\mathcal{R}_\beta$ the set of relations (tables) on β. The attribute set β is called *type of* q and denoted $type(q)$. It is fixed by the query formulation. *POSS*–semantics as defined in the previous section suggests to define semantics of a query q posed against an incomplete database idb as the set of mappings $\{q \mid q : POSS(idb) \rightarrow \mathcal{R}_\beta\}$. Query semantics for incomplete databases can thus be based on query semantics for complete databases. In case of close subsumptions and augmentations, however, it has to be defined how queries are evaluated against relations with *non–existent* values in attributes. For a given incomplete database idb the set $\{q(pdb) \mid pdb \in POSS(idb)\}$ is called set of *possible answers* to q for idb.

In [Lip79] an *upper value* has been defined for selection queries against a so–called object information system. This value corresponds to the union of possible answers and is finite for finite systems. It is not well–suited for the relational data model in general since it can be infinite even for finite databases and too much information about the structure of possible answers can be lost: Consider two relations $R = \{(1, \omega)\}$ and $S = \{(1, 2), (1, 3)\}$ on $\{A, B\}$ assuming ω to represent exactly one missing value. The possible answers to $R[B \neq 1]$ are all singletons $\{(1, c)\}$ with $c \neq 1$; its union is infinite if the domain of B is infinite. The possible answers to the join of R and S are $\emptyset, \{(1, 2)\}$ and $\{(1, 3)\}$. The union $\{(1, 2), (1, 3)\}$ does not reflect these three possibilities.

The *lower value* defined in [Lip79] is the intersection of possible answers. It is a useful approximation to sure information answers for the information system considered in [Lip79] and is often applied to the relational data model (e.g.: [Gra91], [AHV95]). In [IL84] it is used in connection with the discussion of so–called *representation systems*. Such a system exists for a given set of relational operators, a given representation form for relations (single null value or variables), and a fixed semantics if the following holds: Answers can be computed incrementally for arbitrary expressions over the given set of operators without losing information with respect to the *lower value*, i.e. the intersection of possible answers does not change. This means that intermediate results can be represented in a finite form such that no information is lost necessary for the determination of the *lower value* for the complete expression (*truth–preservation*). One of the main results in [IL84] is that for so–called *Codd-tables* (a single null value and completion semantics) such representation systems only exist for the trivial empty set of operators and sets containing projection and/or selection operations with equality as single comparison operator. As a consequence, we cannot expect to find a finite representation for the set of possible answers in general (see also the example above). This negative result holds for *V–tables* (relations with variables) as well if the set of operators contains non-monotone operators ([IL84]). Thus one has to look for approximations to the set of possible answers.

3.2 Sure Information Answers as Approximations

For the following, it is important to note that missing values in close subsumptions and augmentations are interpreted as *value not existent* and treated like constants in the information order. Let us apply the concept of a *lower value* as an approximation to the set of possible answers for relational databases.

Definition: The *sure total answer* to a query q posed against some incomplete database idb is defined as the set $\bigcap_{pdb \in POSS(idb)} q(pdb)$.

This kind of answer is also called *f–information* ([IL84]), *certain answer* ([Gra91]), or *set of sure facts* ([AHV95]). The restriction to total tuples representing sure information makes it more suitable for theoretical investigations than for practical purposes. To give an example, for the query 'R' we get an empty sure total answer if $R = \{(1, \omega)\}$; for '$R[A]$' the answer is $\{(1)\}$.

From a practical viewpoint, partial tuples have to be allowed in answers. Since semantics of an incomplete database is given by a set of complete databases, it is reasonable to consider all partial tuples as sure information having an extension in every possible answer.

Definition: The *set of sure answer tuples* to q posed against idb, written $S(q, idb)$, is defined as follows:

$$S(q, idb) =_{def} \{t \mid t \text{ partial tuple on } type(q) \quad \wedge \\ (\forall pdb \in POSS(idb))(\exists t' \in q(pdb))(t' \geq t)\}.$$

A set of sure answer tuples is always a lower semi–lattice with respect to the information ordering on partial tuples. Since it is uniquely defined by its maximal elements, the following kind of sure answer is of interest:

Definition: The *sure max–answer* to q posed against idb is the set of all maximal elements of $S(q, idb)$.

Obviously, the sure total answer to a query is always a subset of the sure max–answer. Both answers coincide if the set of sure answer tuples does not contain maximal tuples which are partial.

Example 2: Let an incomplete database idb consist of the compatible relations $P = \{(1,2),(1,3)\}$ and $Q = \{(1, \omega)\}$ with ω assumed to represent exactly one missing value (completion semantics). In every possible world to idb the relation for Q consists of exactly one tuple. Consider the query $P - Q$. Either $(1, 2)$ or $(1, 3)$ or both tuples of P qualify for the answer. Hence we get $\{(1, \omega), (\omega, \omega)\}$ as set of sure answer tuples and $\{(1, \omega)\}$ as sure max–answer. The sure total answer is empty. In case of close extension semantics the set of sure answer tuples is empty as well.

Sure max–answers can be enriched by subsumed sure answer tuples in order to get better approximations for possible answers ([Kln99a], [Kln99b]).

Occurrences of null values in sure max–answers have to be distinguished from those in incomplete databases insofar as they do not represent arbitrary values in general (cf. Example 2). If simple efficient evaluation methods such as three–valued logic are applied, the meaning of nulls in the database and in answers is *uniform*, i.e. null values in answers represent arbitrary values. Therefore subsets of the following kind of answer are of special interest:

Definition: Let q and idb be as before. The *sure uni–answer* to q posed against idb is the following subset of $S(q, idb)$:

$$\{t \in S(q, idb) \mid (\forall A \in type(q))(t(A) = \omega \Rightarrow (\forall d \in D)(\exists t' \text{ on } type(q))$$
$$(t'(A) = d \ \wedge \ t' \geq t \ \wedge \ (\exists pdb \in POSS(idb))(t' \in q(pdb))))\}.$$

The sure uni–answer to $P - Q$ from Example 2 is empty since ω in the sure answer tuple $(1, \omega)$ is to be interpreted as 1 or 2.

4 Existential Nulls

4.1 Tautology and Decidability Problems

In the following we assume the *unknown* interpretation for null values and denote occurrences of these values in examples by *unk*. A query against an incomplete database with this kind of missing information may state conditions which are fulfilled or not independent of the replacement of null values by concrete values. Such tautologies and contradictions may be valid independent of the given database or they may hold for a part of the database.

Example 3: Let $R = \{(5, unk, 2), (unk, 1, 3)\}$ be a relation on $\{A, B, C\}$. The query $q = R[A \geq B \vee A < B]$ formulates a condition which is true for tuples of R independent of their values in the attributes A and B.

The condition formulated in the query $q = R[A > 4 \wedge B = 0 \vee C < 3 \wedge B \neq 0]$ is false for the second tuple but true for the first tuple independent of the null value in A and B, resp.

Co–NP–completeness with respect to query complexity has been shown in [Var86]. Further complexity problems have been investigated for different representation forms of incomplete information in [AKG91]. These complexity problems force to look for approximations to all kinds of sure answers.

For close extension semantics not only complexity is the problem: Since a null value represents an arbitrary number of missing values it follows from the undecidability of finite validity in first–order logic (theorem of Trachtenbrot) that sure information answers are not computable for query languages which are as expressive as relational calculus (for a proof see [Kln97]).

4.2 Approximations for Relational Calculi and Algebra

In [Kle38] S.C. Kleene introduced a three–valued logic for reasoning about partial recursive functions. The third truth value represents the indefiniteness of computation, a situation quite similar to the missing knowledge about values. The tables for the connectives *and*, *or*, and *not* are identical to the three–valued logic proposed earlier by J. Łukasiewicz for the investigation of modality:

$\wedge$	T	F	U
T	T	F	U
F	F	F	F
U	U	F	U

$\vee$	T	F	U
T	T	T	T
F	T	F	U
U	T	U	U

$\neg$	T	F	U
	F	T	U

Both logics differ in the definition of implication. Kleene's definition ($U \Rightarrow U$ is equivalent to U) fits well the *unknown* interpretation. Therefore, equivalences of this logic should be valid when three–valued logic is used for relational query languages.

In [Cod79] the above definitions are used for the evaluation of calculus queries under completion semantics. Comparison expressions are evaluated to *unknown* if the null value occurs in the arguments. Quantifiers are assumed to behave like iterated *or* and *and*. The monotonicity property of three–valued logic guarantees that tuples in results are always sure answer tuples: If some tuple qualifies for the answer then it qualifies even if at evaluation some occurrences of *unknown* are replaced by *true* or *false*. This means that *true* (and *false*) results are independent of null values. One problem has to be observed, however: If the identification of tuples is stated indirectly, i.e. without use of comparison expressions, as in $(x)/Rx \wedge Sx$, then the calculus interpretation has to avoid the identification of independent partial tuples. Tuple identification in set operations in connection with the removal of duplicates is regarded in [Cod79] as "an operation at a lower level of detail than equality testing in the evaluation of retrieval conditions". This view is exactly what we find in SQL where three–valued logic is used for the evaluation of WHERE–clauses but independent occurrences of the null value may be identified in set operations ([ISO92]). As a consequence, set operations may yield results not representing sure information.

If operations of relational algebra shall be defined in accordance with possible world semantics, the independence of different occurrences of null values in a database has to be guaranteed. In [Bis83] the need to change completion semantics into close extension semantics in results of operations is argued. Tuples in results are classified as *sure* or *maybe* depending on their status with respect to evaluation. Definitions for relational algebra operations are given and proven to guarantee the correct status of tuples in results. Expressions of relational algebra, however, are not considered. On the basis of these definitions and the proposal in [Cod79] to consider sure and maybe versions of operations, a simple evaluation method for relational algebra expression can be developed.

For every operation, a sure and a maybe version are defined: The sure version includes only tuples in the result fulfilling for sure the condition of the operation; the maybe version includes in addition every tuple which may not be excluded from the result (notice the difference to the definitions in [Cod79] where the maybe–result only includes these additional tuples). As an example consider the two versions of difference: Let R, S be relations on the same set of attributes; $R -_s S =_{def} \{t \mid t \in R \wedge \neg(\exists t' \in S)(t' \Delta t)\}$, $R -_m S =_{def} R - \{t \mid t \in S$ and total$\}$ where "$-$" is the ordinary set difference.

For the evaluation of an arbitrary expression e sure and maybe versions of op-

erations are chosen depending on the "level of negation". For operations on the outermost level of e sure versions are used; for a difference operation the operations at the outermost level of the expression representing the subtrahend are evaluated applying their maybe–version. This proceeding is iterated switching from maybe to sure and vice versa.
Example: For $R - ((S - R[A = 2]) \cup T)$ we get $R -_s ((S -_m R[A = 2]_s) \cup_m T)$.

We call this method *switch–evaluation*. It can be shown that applying switch–evaluation to relational algebra expressions and three–valued logic to relational calculus queries leads to the same results for algebra expressions and calculus queries being equivalent in case of complete databases ([Kln97]). This holds for completion as well as for close extension semantics. Thus the well–known equivalence theorem can be lifted to incomplete databases. Answers are always subsets of the sure uni–answer.

In order to approximate sure max–answers without loss of efficiency, extended definitions of algebra operations and multivalued–logics for calculus queries can be introduced ([Kln99b]). For all kinds of answers, approximations can be improved without loss of efficiency by exploiting identities of unknown values established with evaluation.

4.3 Literature Review

Many proposals for handling the *unknown*–interpretation of missing values present definitions of algebraic operators without giving declarative semantics for queries. Here we discuss some of them with respect to sure information in answers.

The approach of [Cod79] does not guarantee sure information in answers because null values are handled in set operations like ordinary values. Let $R = S = \{(1, unk)\}$; then $R \cap S$ is defined as $\{(1, unk)\}$. The combination of maybe–results for the evaluation of expressions may lead to results which are not sound in any possible world. Consider the expression $(R - R[B = 2]) \cap (R - R[B <> 2])$. We get R as result; under completion semantics the result is empty in every possible world.

In [GZ88], different types of null values are considered among them *unknown* with completion semantics and *open* with close subsumption semantics. Symbolic equality is used in set operations such that the preceding remarks hold as well. For *open* nulls, undecidability follows from undecidability in case of close extension semantics.

The weakness of "truth-preserving extension" based on representation systems (see Section 3.1) shows up in definitions of extended operators given in [LeLo99] for null values with completion semantics. Consider the definition of extended difference given there: $R -_e S =_{def} \{t \mid t \in R \wedge \neg(\exists u \in S)(u \leq t \vee t \leq u\}$. Let $R = \{(1, unk)\}, S = \{(unk, 2)\}$. Then $R -_e S = \{(1, unk)\}$ since neither $\{(unk, 2)\} \leq \{(1, unk)\}$ nor $\{(1, unk)\} \leq \{(unk, 2)\}$. In order to restrict the result to sure information, compatibility has to be excluded for u and t (cf. our definition of the sure version of difference). Similar comments hold for extensions proposed in connection with the nested relation data model (e.g. [RKS89]).

A completely different approach to the problem of inefficiency in connection with the evaluation of queries against incomplete databases has been taken in [Lak89]. Query evaluation is based on intuitionistic logic and it is claimed that the missing of the tautology problem in this logic allows to compute complete answers efficiently. Of course, answers have a special meaning in this approach and it can be shown that they do not always represent sure information in the model–theoretic sence ([Kln97]).

It has been indicated an advantage of the proof–theoretic approach over to the model–theoretic approach that in case of incompleteness the introduction of new logics can be avoided ([Rei86]). However, a closer look at the evaluation methods proposed in [Rei86] and [Var86] shows that both give the same result as an appropriate interpretation of calculus formulas using three–valued logic for databases with variables as null values ([Kln98]). This means that peculiarities connected with three–valued logic (e.g.: the equivalence $A = B \wedge B = C \equiv A = B \wedge A = C$ does not always hold) are present as well.

5 No Information Nulls and SQL

5.1 Semantics

[Zan84] proposes to base the definition of operators for relations with *no information* nulls on equivalence classes of relations on extensible attribute sets. [Kel86] shows that there is no way to define relational algebra operators on equivalence classes of this kind such that the definitions are in accordance with ordinary set theory. The definitions of relational operators in [Zan84] do also not guarantee sure information in results. As an example take the definition of difference (missing values are represented by *null* in this section): $R\tilde{-}S =_{def} \widehat{\{t \mid t \in R \text{ and } (\forall s \in S)(\neg(s \geq t))\}}$. Here $\widehat{\{..\}}$ denotes the equivalence class for $\{..\}$. For $R = \{(1,2)\}$ and $S = \{(1, null)\}$ we get $R\tilde{-}S = \widehat{\{(1,2)\}}$.

Possible world semantics for databases with *no information* nulls has to take care of a representation problem arising in connection with the closed world assumption.

Example 4: Consider the following relations with information on the names of invited guests of a party and their telefon number in GUESTS and the names of some guests together with the name of an accompanying person in COMPANY. For Frank we know that he comes alone whereas for Petty we know nothing with respect to her company.

GUESTS

Gname	Tel#
Tim	346537
Petty	660890
Frank	730286

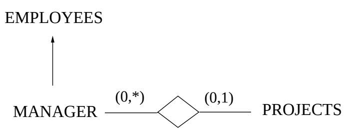
COMPANY

Gname	Cname
Tim	Anna
Petty	null

If we consider all possible worlds then in one of these worlds the information is modeled that Petty is not accompanied. This means that in this possible world the null value is not replaced by a data value but it is interpreted as *value* *not existent*. Hence the information on the company of Frank is represented differently from the information on the company of Petty. This may lead to inconsistencies with respect to query evaluation. Consider the following SQL–query:

```
SELECT * FROM GUESTS x WHERE EXISTS (SELECT * FROM COMPANY y
                                     WHERE y.Gname = x.Gname)
```

The SQL–answer to this query is $\{$(Tim,346537), (Petty,660890)$\}$. It does not represent sure information if we think of a formulation asking for guests accompanied by someone. A first idea is to specify $POSS$–semantics depending on a given query and to remove partial tuples in possible worlds if there are conflicts with the CWA. The next example demonstrates that this proceeding does not work in general without additional schema information.
Example 5: Consider the following relations modeling information on employees and project managers as indicated by the entity/relationship–diagram.



EMPLOYEES

Name	Tel#
Ben	375
Jane	897
Peter	993

PROJMAN

Name	Proj#
Ben	P3
Jane	null

Look at the following query having the same structure as the query above:

```
SELECT * FROM EMPLOYEES x WHERE EXISTS (SELECT * FROM PROJMAN y
                                        WHERE y.Name = x.Name)
```

Now the intension of the query could be to get the data of all employees being managers without requiring that they actually lead projects. Hence in the possible world corresponding to that one mentioned in the preceding example no tuple has to be discarded for solving conflicts with the CWA. This means that from the structure of the relations and queries alone we cannot derive a satisfying definition of sure information answers. Obviously, it is necessary to know more about *concepts* modeled in schemas and addressed in queries. In the GUESTS/COMPANY–example, a single concept "guest comes in company of a person" is modeled, in the EMPLOYEES/PROJMAN–example two concepts are modeled: "an employee e is a manager" and "a manager e controls a project p". The standard relational data model does not allow to address such concepts. An appropriate proposal to extend the model by specifying sets of attributes as units of interest can already be found in [Mai83] (see also [Sci79] and [IL82]). There the term *object* is used for characterizing allowable schemes for subtuples in relations with tuples padded out by placeholders. We refine this proposal by distinguishing mandatory and optional attributes in order to be able to differentiate values which are necessary for a combination of values to "make sense" from values which may be considered as supplementary information.
Definition: A *concept* for a relation type is a subset of its set of attributes with elements classified either as *mandatory* or as *optional*.
Example 6: Consider the relation type PARTY(Gname, Tel#, Cname, Contribution) for modeling information about names of guests, their telefon number,

the name of the person in their company, and their contribution to the buffet. The following are reasonable concepts for this type: {(Gname,m), (Tel#,o)}, {(Gname,m), (Cname,m)}, and {(Gname,m), (Contribution,m)}.
For the relation type PROJMAN(Name, Proj#) from above we could choose: {(Name,m)} and {(Name,m), (Proj#,m)}.

Given a non–empty set of concepts for a relation type we can now define the set of tuples "making sense".

Definition: Let (RT, α) be a relation type with set of concepts $\mathcal{C}$. A partial tuple t on α is *allowed with respect to* $\mathcal{C}$ $\Leftrightarrow_{def}$

- t is not totally undefined.
- If t is defined in an attribute A then there is at least one concept in $\mathcal{C}$ containing A such that t is defined in all mandatory attributes of this concept.

A relation R on α is *allowed with respect to* $\mathcal{C}$ $\Leftrightarrow_{def}$ every tuple of R is allowed with respect to $\mathcal{C}$.

These definitions permit to move semantics at the level of syntax.

Definition: Let (RT, α) be a relation type with set of concepts $\mathcal{C}$ and let R be an incomplete relation on α. A relation R' is called *possible relation* to R with respect to $\mathcal{C}$ $\Leftrightarrow_{def}$ R' can be obtained from an augmentation (close subsumption) R'' of R by discarding all tuples which are not allowed.

Augmentations are appropriate if null values are not allowed in key attributes; otherwise, close subsumptions have to be considered. Since possible worlds may contain values with *non–existent* interpretation, it has to be fixed how these values are treated in query evaluation. Taking the view that it makes no sense to compare "nothing" with a data value ([Vas79]) the following approach seems to be appropriate: Consider relational calculus as a reference language; make use of three–valued logic as in Section 4 with *non–existent* entries in attributes treated like *unknown* values.
If both operands in a comparison expression are *non–existent* values, a Boolean value could be assigned to the expression. There are, however, also arguments to determine that a non–existent value does not satisfy any relational expression ([Zan84]). Furthermore, for such a refinement the addressing of concepts in queries has to be taken into account which is beyond the scope of this paper.

5.2 Peculiarities in SQL

In the following, we restrict our attention to *SELECT...FROM...WHERE*–clauses in SQL. Problems in connection with the duplicate removal rule in set operations applied in SQL have already been addressed in Section 4.

The SQL–standard specifies a null value as "a special value, or mark, that is used to indicate the absence of any data value" ([ISO92]), a characterization fitting well the *no information* interpretation. If a null value occurs in a comparison expression, the expression is always evaluated to *unknown*. For the evaluation of WHERE–clauses three–valued logic is applied. In the result of a FROM–clause only tuples are retained for which the where–clause evaluates to *true*. This switch from three–valued to Boolean logic may lead to answers not representing sure information in the same way as the approaches discussed in Section 4.

Example 7 ("Maybe and maybe may be not maybe")**:** Let $R = \{(1, null)\}$ be a relation on $\{A, B\}$.

```
SELECT x.A FROM  R x WHERE NOT EXISTS (SELECT * FROM  R y WHERE y.B = 2)
                       AND NOT EXISTS (SELECT * FROM  R z WHERE z.B <> 2)
```

The SQL–answer to this query is $\{(1)\}$ since in both subqueries the WHERE–clauses evaluate to *unknown*. Obviously, the answer is empty if *null* is replaced by any value of the domain of B.

SQL–semantics allows a programmer to interpret null values by chosing the appropriate query formulation, i.e. to get results fitting the *unknown* or the *non–existent* interpretation.

Example 8: Let tuples in the following relation represent information on orders with a unique number, the date of their entry, and the identification number of the employee working on it.

ORDER

Ordno	Date	Works_on
1	03-nov	null
3	03-nov	E2
4	04-nov	null

Consider the query 'Get the data of each order o such that there is no employee working on o and on further orders with the same date of entry as o.'
Assuming *non–existent* interpretation for nulls ("no employee is working on the order") the appropriate answer is the relation ORDER. We get it by chosing the following formulation:

```
SELECT * FROM ORDER x WHERE NOT EXISTS (SELECT * FROM ORDER y WHERE
  y.Date = x.Date AND y.Ordno <> x.Ordno AND y.Works_on = x.Works_on)
```

Assuming *unknown* interpretation, a sure information answer should consist only of the third tuple. The following formulation is appropriate:

```
SELECT * FROM ORDER x WHERE Works_on <> ALL (SELECT Works_on FROM
            ORDER y WHERE y.Date = x.Date AND y.Ordno <> x.Ordno)
```

This formulation is suited also if the *no information* interpretation is assumed.

In general, equivalences valid in Kleene's three–valued logic do not hold in SQL. Applying the equivalence of $A \wedge \neg B \Rightarrow \neg C$ and $C \wedge \neg B \Rightarrow \neg A$, for example, to the second formulation above we get a formulation equivalent to the first formulation:

```
SELECT * FROM ORDER x WHERE x.Date <> ALL (SELECT y.Date FROM
            ORDER y WHERE y.Works_on = x.Works_on  AND y.Ordno <> x.Ordno)
```

By making use of the IS NULL predicate of SQL, it is always possible to choose a query formulation having a subset of the sure uni–answer as result for any of the above interpretations. How this can be done systematically by query transformation will be described in the following.

5.3 Modification of SQL Queries

In Section 4 we proposed the *switch–algebra* for the *unknown*–interpretation. Based on that idea, transformation rules can be developed for SQL such that transformed queries have sure information answers if concepts are taken into account appropriately.
Let an arbitrary SELECT...FROM...WHERE–clause w be given. Because of the monotonicity property of three–valued logic, the evaluation of the WHERE–clause generates sure information provided that all Boolean values used for the evaluation represent sure information. For a negated subformula $NOT(w')$ of w this means that w' has to evaluate to *true* if it cannot be excluded that there is a possibility to replace null values such that w' has the value $true$. The possible loss of the monotonicity property by the transition from three–valued logic to Boolean logic in predicates has to be avoided by appropriate modification of the predicates. In analogy to the switch–algebra, $sure$ and $maybe$ versions of predicates are needed. $unknown$ as result of a predicate or of the WHERE–clause of a predicate has to be turned into a $false$ or $true$ depending on the kind of result needed. The switch from $sure$ to $maybe$ and vice versa is caused by negation and universal quantification.
Concepts are taken into account by deleting at the "sure level" all non–allowed tuples in the corresponding tables. An appropriate conditional expression can be formulated with the help of the IS NULL predicate. At the "maybe level" no tuple has to be deleted.
As an example, let us consider the sure version of the *quantified comparison predicate* with ALL as quantifier (for the maybe version and other predicates see [Kln94]). Its general form is $v\Theta ALL(S)$ where v is a row value expression, S stands for a suitable subquery, and Θ is a comparison operator. To get a sure result, the maybe result is needed for S. There are two critical cases to consider:
1) v evaluates to *null*.
In this case, the SQL answer is *true* if S evaluates to $\emptyset$ and *unknown*, otherwise. *unknown* can be changed to *false* by adding AND (v IS NOT NULL OR NOT EXISTS(S)) to the predicate.
2) *null* occurs in the result of the evaluation of S (e.g.: $5 \leq$ ALL{7,8,null}).
false instead of the possible result *unknown* has to be guaranteed. For simplicity, let S = SELECT B FROM ... WHERE *cond*. Add AND NOT EXISTS(S') to the predicate where S' = SELECT B FROM ...WHERE *cond* AND B IS NULL.
The application of this rule to the third formulation in Example 8 gives:

```
SELECT * FROM ORDER x WHERE x.Date <> ALL (SELECT y.Date FROM ORDER y
         WHERE (y.Works_on = x.Works_on OR y.Works_on IS NULL
                OR x.Works_on IS NULL)  AND y.Ordno <> x.Ordno)
-         AND x.Date IS NOT NULL
-         AND NOT EXISTS (SELECT y.Date FROM ORDER y WHERE
-           y.Ordno <> x.Ordno AND y.Date IS NULL)
```

With {(Ordno,m),(Date,m)} and {(Ordno,m),(Works_on,m)} as concepts, the condition AND (y.Ordno IS NOT NULL AND y.Date IS NOT NULL OR y.ORDNO IS NOT NULL AND y.Works_on IS NOT NULL) has to be added to the WHERE–clauses of both subqueries. The conditions in the lines marked by "–" are redundant and can be removed if in the attribute 'Date' null values are not allowed. Simplifications of this kind allow to reduce the size of the generated formulation in many cases.

Instead of answers representing sure information, "maybe" answers can be produced by starting with maybe–versions of predicates (including simple comparison predicates) at the outer level. However, these "maybe" answers may contain information not implied by any possible world because of contradictions which are not detected.

6 Conclusions

There is much controversy about the way incomplete information should be represented and processed by relational database systems. We suggest that systems should provide interfaces avoiding the confrontation of the user with the intrinsic complexity of incompleteness. Different kinds of answers with well–defined non–procedural semantics should be requestable at different levels of completeness allowing users to explore incomplete information at costs they are willing to spend (cf. [KW85]). To have answers with different semantics and increasing levels of completeness at hand, may prevent users from wrong conclusions.

The use of three–valued logic has been criticized for producing incomplete results ([Grt77]). However, if this logic is carefully applied to avoid unallowed identification of missing information, it provides a good foundation for efficient computation of answers representing sure information. Evaluation methods based on this logic or their corresponding extended operators in relational algebra can be refined with moderate increase of costs such that better approximations to sure information answers can be achieved. As we have seen, some proposals in the literature fail to guarantee sure information in answers so that the interpretation of answers becomes problematic.

In this paper we considered standard languages for the formulation of queries since this is the current reality we have to cope with. Another approach is to look for new languages better suited for the formulation of requests against databases with incomplete information ([Lip81], [Lib98]). This could help to get a common framework for dealing with the many facets incompleteness shows.

References

[AHV95] S. Abiteboul, R. Hull, and V. Vianu: Foundations of Databases. Addison–Wesley, Reading, 1995

[AKG91] S. Abiteboul, P. Kanellakis, and G. Grahne: On the representation and querying of sets of possible worlds. Theor. Comp. Science 78, pp. 159–187, 1991

[Bis83] J. Biskup: A foundation of Codd's relational maybe-operations. ACM Trans. on Database Systems 8 (4), pp. 608–636, 1983

[Cod72a] E.F. Codd: Further normalization of the database relational model. In: Data Base Systems. Current Computer Science Symposium 6 (R. Rustin, ed.), Prentice Hall, Englewood Cliffs, pp. 33–64, 1972

[Cod72b] E.F. Codd: Relational completeness of data base sublanguages. In: Data Base Systems. Current Computer Science Symposium 6 (R. Rustin, ed.), Prentice Hall, Englewood Cliffs, pp. 65–98, 1972

[Cod79] E.F. Codd: Extending the database relational model to capture more meaning. ACM Trans. on Database Systems 4 (4), pp. 397–434, 1979

[Cod86] E.F. Codd: Missing information (applicable and inapplicable) in relational databases. ACM SIGMOD RECORD 15 (4), pp. 53–77, 1986

[Dat86] C.J. Date: Null values in database management. In: Relational Database: Selected Writings, Addison–Wesley, Reading, 1986

[Dat90] C.J. Date: Relational Database Writings 1985–1989. Addison-Wesley, Reading, 1990

[DD98] C.J. Date and H. Darwen: Foundation for Object/Relational Databases. Addison Wesley, Reading, 1998

[Ges90] G.H. Gessert: Four valued logic for relational database systems. ACM SIGMOD Record 19 (1), pp. 29–35, 1990

[GM90] O.N. Garcia and M. Moussavi: A six–valued logic for representing incomplete knowledge. Proc. IEEE 20th Int. Symp. on Multiple–valued Logic (G. Epstein, ed.), Charlotte, pp. 110–114, 1990

[Gra91] G. Grahne: The Problem of Incomplete Information in Relational Databases. LNCS 554, Springer–Verlag, Berlin, 1991

[Grt77] J. Grant: Null values in relational data base. Inform. Proc. Letters 6 (5), pp. 156–157, 1977

[GZ88] G. Gottlob and R. Zicari: Closed world databases opened through null values. Proc. 14th VLDB Conf., Los Angeles, pp. 50–61, 1988

[IL82] T. Imielinski and W. Lipski: A systematic approach to relational database theory. Proc. ACM SIGMOD Conf., Orlando, pp. 8–14, 1982

[IL84] T. Imielinski and W. Lipski: Incomplete information in relational databases. Journal of the ACM 31 (4), pp. 761–791, 1984

[ISO92] International Standard ISO/IEC 9075, third edition. Genève, 1992

[Kel86] A.M. Keller: Set–theoretic problems of null completion in relational databases. Inf. Proc. Letters 22, pp. 261–265, 1986

[Kle38] S.C. Kleene: On a notation of ordinal numbers. The Journal of Symbolic Logic 3, pp. 150–155, 1938

[Kln94] H.-J. Klein: How to modify SQL queries in order to guarantee sure answers. ACM SIGMOD Record 23 (3), pp. 14–20, 1994

[Kln97] H.-J. Klein: Sure and possible answers to queries for relational databases with partial relations. Technical Report 9802, Inst. f. Informatik u. Prakt. Mathem., Univ. Kiel, 194 pages, 1997 (in German)

[Kln98] H.-J. Klein: Model Theoretic and Proof Theoretic View of Relational Databases with Null Values: a Comparison, in: M.H. Scholl et al. (Eds.), 10. GI-Workshop "Grundlagen von Datenbanken", Konstanzer Schriften in Mathematik u. Informatik, Nr. 63, Univ. Konstanz, pp. 57–61, 1998

[Kln99a] H.-J. Klein: On the use of marked nulls for the evaluation of queries against incomplete relational databases. In: Fundamentals of Informations Systems (T. Polle, T. Ripke, and K.-D. Schewe, eds), Kluwer Academic Publ., Boston, pp. 82–99, 1999

[Kln99b] H.-J. Klein: Efficient algorithms for approximating answers to queries against incomplete relational databases. In: Proc. KRDB'99 (E. Franconi and M. Kifer, eds.), Linköping, pp. 26–30, 1999

[KW85] A.M Keller and M.W. Winslett Wilkins: On the use of an extended relational model to handle changing information and nulls. IEEE Trans. on Soft. Eng., SE–11 (7), pp. 620–633, 1985

[Lak89] V.S. Lakshmanan: Query evaluation with null values: how complex is completeness?. Proc. 9th Conf. on Found. of Software Technologies and Theor. Comp. Science, Bangalore, LNCS 405, Springer–Verlag, Berlin, pp. 204–222, 1989

[LeLi86] N. Lerat and W. Lipski: Nonapplicable nulls. Theoretical Computer Science 46, pp. 67–82, 1986

[LeLo99] M. Levene and G. Loizou: A Guided Tour of Relational Databases and Beyond. Springer–Verlag, London, 1999

[Lib98] L. Libkin: A semantics based approach to design of query languages for partial information. In: Semantics in Databases (B. Thalheim, L. Libkin, eds.), LNCS 1358, pp. 170–208, 1998

[Lip79] W. Lipski: On semantic issues connected with incomplete information databases. ACM Trans. on Database Systems 4 (3), pp. 262–296, 1979

[Lip81] W. Lipski: On databases with incomplete information. J. of the ACM 18 (1), pp. 41–70, 1981

[LS90] K.C. Liu and R. Sunderraman: Indefinite and maybe information in relational databases. ACM Trans. on Database Systems 15 (1), pp. 1–39, 1990

[Mai83] D. Maier: The Theory of Relational Databases. Computer Science Press, Rockville, 1983

[MS97] A. Motro and P. Smets (eds.): Uncertainty Management in Information Systems. Kluwer Academic Publ., Boston, 1997

[MUV84] D. Maier, J.D. Ullman, and M.Y. Vardi: On the foundations of the universal relational model. ACM Trans. on Database Systems 9 (2), pp. 283–308, 1984

[OO93] A. Ola and G. Ozsoyoglu: Incomplete relational database models based on intervals. IEEE Trans. on Knowledge and Data Eng. 5 (2), pp. 294–308, 1993

[Rei78] R. Reiter: On closed world databases. In: Logic and Databases (H. Gallaire and J. Minker, eds.), Plenum Press, New York, pp. 55–76, 1978

[Rei86] R. Reiter: A sound and sometimes complete query evaluation algorithm for relational databases with null values. J. of the ACM 33 (2), pp. 349–370, 1986

[RKS89] M.A. Roth, H.F. Korth, and A. Silberschatz: Null values in nested relational databases. Acta Informatica 26, pp 615–642, 1989

[Sci79] E. Sciore: Improving semantic specification in a relational database. Proc. ACM SIGMOD Conf., Boston, pp. 170–178, 1979

[Var86] M.Y. Vardi: Querying logical databases. Journal of Comp. and System Sciences 33, pp. 142–160, 1986

[Vas79] Y. Vassiliou: Null values in data base management: a denotational approach. Proc. ACM SIGMOD Conf., Boston, pp. 162–169, 1979

[Yue91] K. Yue: A more general model for handling missing information in relational databases using 3–value logic. ACM SIGMOD 20 (3), pp. 43–49, 1991

[Zan84] C. Zaniolo: Database relations with null values. J. of Comp. and System Sciences 28, pp. 142–166, 1984

[ZP97] E. Zimányi and A. Pirotte: Imperfect information in relational databases. In: [MS97], pp. 35–87, 1997