# Efficient Algorithms for Approximating Answers to Queries Against Incomplete Relational Databases

**Hans–Joachim Klein**
Institut für Informatik und Praktische Mathematik, Universität Kiel, 24098 Kiel, Germany

## 1 Introduction

Computing nontrivial answers to queries against relational databases with 'unknown' values in attributes is directly related to the tautology problem for Boolean expressions ([Abiteboul *et al.*1991]), i.e. we face a co–NP–complete problem when we try to evaluate incomplete information. Things are even worse if unknown values are represented by a single special symbol in the database and if every occurrence of this symbol stands for a non–empty set of unknown values. With these assumptions answers are not computable in general.

There are numerous proposals for dealing with incomplete information by using modified relational operators or special logics or by transforming expressions in a specific way (for a recent survey see [Zimányi and Pirotte1997]). The major drawback of many of these proposals is that a user has to know the evaluation method in order to be able to interpret answers semantically. Problems connected with this proceeding show up in SQL where the combination of Boolean logic and a 3–valued logic results in query semantics being sometimes not in accordance with intuition and being in contradiction to equivalences valid in Boolean logic or Kleene's three–valued logic, for example. A more suitable approach seems to be the specification of query semantics independent of query evaluation, to look for appropriate forms for the representation of answers, and to develop algorithms for query evaluation with respect to the different settings.

The main contributions of this paper are the introduction of denotational definitions of sure answers allowing unknown values in attributes, the proposal of a new 5–valued logic for improving the approximation of answers, and a comparison of efficient evaluation methods for the proof theoretic and the model theoretic view.

## 2 Preliminaries

We restrict our attention to missing attribute values of type 'value existent but at present unknown'. Two forms for modelling incomplete information of this kind in a relational database are often considered: A single special symbol (*null value*) can be used to represent missing values or every occurrence of a missing value can be denoted by a unique symbol (*marked null values*). We start with the first possibility and review some basic notions (cf. [Maier1983]). For simplicity we do not take integrity constraints into account.

Let $D$ be a countably infinite set of atomic values, called *domain*, and let $\omega$ be a special symbol which is distinct from every element of $D$. A *partial database relation*, also called *$\omega$–relation*, is a finite subset of $(D \cup \{\omega\})^k, k \in \mathbb{N}$; its elements are called *$\omega$–tuples*. It is convenient to give names, called *attributes*, to all columns of a relation and to consider a tuple as a function mapping the attributes to elements of $D \cup \{\omega\}$. An *incomplete database* over some database schema $\sigma = \{(id_1, \alpha_1), ...., (id_m, \alpha_m)\}$, $id_i$ unique identifier, $\alpha_i$ set of attributes, $i = 1, ..., m$, is a set of $m$ partial relations with one relation for every relation type $(id_i, \alpha_i)$ of $\sigma$.

As a consequence of the meaning of 'unknown', the semantics of a partial database relation $P$ can be defined by a set of total database relations (no occurrences of $\omega$) called *possible relations* to $P$. Since relations are sets, two kinds of semantics are of special interest under the closed world assumption. The first one assumes occurrences of $\omega$ to represent exactly one unknown value (appropriate, for example, if unknown values are not allowed in key attributes), the other one assumes occurrences of $\omega$ to represent an arbitrary non–empty set of unknown values (appropriate, if duplicates may occur in relations as result of update or projection operations). In the first case, semantics of a partial relation $P$ is given by the set of all total relations obtained from $P$ by replacing every occurrence of $\omega$ by exactly one ordinary value (*completions* of $P$). In the second case, every occurrence of $\omega$ can be replaced by one or more ordinary values (*close extensions* of $P$).

**Example 1:** For the $\omega$–relations $P$ and $Q$ below with attributes ranging over the natural numbers two completions $(R_1, R_2)$ and two close extensions $(R_3, R_4)$, respectively, are shown:

$P$

| A | B | C |
|---|---|---|
| 1 | 1 | $\omega$ |
| 1 | 2 | $\omega$ |
| 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 |

$Q$

| A | B | C |
|---|---|---|
| $\omega$ | 1 | 3 |
| $\omega$ | 1 | $\omega$ |
| 2 | 1 | $\omega$ |

$R_3$

| A | B | C |
|---|---|---|
| 1 | 1 | 3 |
| 3 | 1 | 4 |
| 2 | 1 | 1 |
| 2 | 1 | 3 |

$R_4$

| A | B | C |
|---|---|---|
| 2 | 1 | 3 |
| 2 | 1 | 4 |

For the following definitions it is not necessary to distinguish both kinds of semantics. Thus, we simply denote the set of possible relations for an $\omega$–relation $P$ by $POSS(P)$. For an incomplete database $idb$ we use the notation $POSS(idb)$ to denote the set of *possible databases* resulting from $idb$ by replacing every $\omega$–relation $P$ by an arbitrary element of $POSS(P)$.

There is an ordering induced on $\omega$–tuples over the same set of attributes by the natural information order $d > \omega$ on $D \cup \{\omega\}$. An $\omega$–tuple $p'$ *subsumes* (is *more informative* than) an $\omega$–tuple $p$, written $p' \geq p$, if the value of $p$ in every attribute $A$ with $p(A) \in D$ is identical to the value of $p'$ in $A$ (e.g.: $(1,2) \geq (1,\omega)$). If $p' \geq p$ and $p'$ is total then $p'$ is called an *extension* of $p$. By $POSS(p)$ we denote the set of all extensions of $p$.

The use of marked null values for denoting occurrences of missing values can be formalized as following ([Imielinski and Lipski1984]): Let $V$ be a countably infinite set of *variables* such that $V \cap D = \emptyset$.
A *$V$–tuple* $t$ on $\alpha$ is a mapping $t : \alpha \rightarrow D \cup V$. A *$V$–relation* on $\alpha$ is a finite set of $V$–tuples on $\alpha$. An *incomplete database* over a database schema $\sigma = \{(id_1, \alpha_1), ...., (id_m, \alpha_m)\}$ is a set of $m$ $V$–relations with one relation for each relation type of $\sigma$. A *valuation* $\nu$ for a given set of variables $V$ is a mapping $\nu : V \rightarrow D$. To simplify notation, $\nu$ is extended to $D$ by putting $\nu(d) = d$ for every $d \in D$. $\nu$ naturally extends to $V$–tuples, $V$–relations, and incomplete databases.

## 3 Different kinds of answers

In case of total databases it is convenient to consider a query $q$ as a mapping from a database to a finite structure which may be a database relation or a table of a type fixed by $q$. In this paper we assume queries to be formulated in relational calculus and answers to be represented by database relations. By *type(q)* we denote the set of attributes of an answer relation to $q$.

Since semantics of an incomplete database is given by a set of total databases, semantics of a query can be defined as the set of answers obtained by evaluating the query in these total databases. We concentrate here upon databases with $\omega$–relations.
In the following, let $q$ denote a query for some database schema $\sigma$ and $idb$ an arbitrary incomplete database over $\sigma$. The semantics of $q$ posed against $idb$ is defined as the set $\{q(pdb) \mid pdb \in POSS(idb)\}$. Each element $q(pdb)$ is called a *possible answer* to $q$ in $idb$.

The problem with this set is that it can be very large or even infinite so that we have to look for finite representations. In [Imielinski and Lipski1984] it is shown that in general we cannot expect a single relation as finite representation. Therefore we have to be content with approximations. The following definition is often used in the literature; it was introduced as 'lower value' in [Lipski1979]:
**Definition:** The *sure total answer* to $q$ posed against $idb$ is defined as the set $\bigcap_{pdb \in POSS(idb)} q(pdb)$.

In [Lipski1979] this definition is used in connection with an object information system and selection expressions as queries. Answers are restricted to those objects which are contained in every possible answer. With respect to relational databases and more general queries sure total answers are not very useful: If we ask for the content of a database relation, for example, we do not get the relation as result if it is partial since all partial tuples are eliminated ($D$ has more than one element). In the following, we introduce other kinds of sure answers taking partial tuples in answers into account.

The information represented by sure total answers is valid for all possible answers. The question arises how partial tuples can be used to represent additional information valid for all possible answers. Since semantics of a partial relation in the database is given by the set of its completions or close extensions, respectively, it is reasonable to consider all those 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:

$$S(q, idb) =_{def} \{p \mid \ p \ \omega\text{–tuple on } type(q) \wedge (\forall pdb \in POSS(idb))(\exists t \in q(pdb))(t \geq p)\}.$$

A set of sure answer tuples is always a lower semi–lattice with respect to the information ordering on $\omega$–tuples. It is uniquely defined by its maximal elements. Therefore 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 consist of the database relations $P = \{(1,2),(1,3)\}$ and $Q = \{(1,\omega)\}$ on attribute set $\{A, B\}$ with $\omega$ assumed to represent exactly one missing value (completion semantics). In every possible database to this incomplete database the total relation for $Q$ consists of exactly one tuple. Consider the query $(x)/Px \wedge \neg(\exists y : Qy)(y.B = x.B)$. 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. Note that in case of close extension semantics the set of sure answer tuples is empty.

Sure max–answers are sometimes a rather coarse approximation to the set of possible answers: To construct a possible answer from the answer relation it may be necessary to add further tuples to a close extension of the answer relation. A better approximation can often be obtained by adding a minimal number of further elements of $S(q, idb)$ such that in case of $S(q, idb) \neq \emptyset$ every possible answer becomes an element of the set $C$ of close extensions of the answer relation and there is no other relation having this property whose set of close extensions is a subset of $C$.

For unknown values in incomplete databases we assume no constraints to be given, i.e. occurrences of $\omega$ represent arbitrary values or sets of values of the domain $D$. In sure answer tuples, however, there may be constraints for occurrences of $\omega$ as demonstrated in the example above. Therefore the semantical interpretation of answer relations has to be fixed appropriately by specifying how possible answers can be constructed from them.

If a uniform meaning of occurrences of $\omega$ in databases and answers shall be guaranteed, we have to make a suitable selection of elements from the set of sure answer tuples. For databases we assume all occurrences of $\omega$ to be independent. In tuples of an answer relation there may be interdependencies because of the structure of the query. We distinguish two kinds of so–called *uni-answers* (*uni* for 'uniform'). The first form assumes semantics of answer tuples to be the same as semantics of tuples in databases, i.e. every extension of an answer tuple is contained in a possible answer.

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

$$\{p \in S(q, idb) \mid (\forall t' \in POSS(p))(\exists pdb \in POSS(idb)) \\ (t' \in q(pdb))\}.$$

Because of possible dependencies between values in tuples of answers this definition is too restrictive in general. The following simple example demonstrates the problem.

**Example 3:** Let $R = \{(1,\omega)\}$ be a database relation in an incomplete database and $(x,y)/Rx \wedge Ry$ a query against the database. The set of sure answer tuples to the query is the set of tuples subsumed by $(1,\omega,1,\omega)$. The strong sure uni–answer is empty since in each possible answer the second and the forth component of the answer tuple are identical. In $POSS(1,\omega,1,\omega)$, however, there are also tuples with different values in these components.

The following kind of answer allows semantics of answer tuples to be different from semantics of tuples in databases; the meaning of occurrences of $\omega$, however, is retained, i.e. for every occurrence of $\omega$ in an answer tuple $p$ there is an extension of $p$ with an arbitrary ordinary value in the corresponding attribute and this extension is a tuple of a possible answer.

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

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

With this definition we get $\{(1,\omega,1,\omega)\}$ as answer to the query in Example 3.

In analogy to the definitions given for databases with $\omega$–relations, the notions of a set of sure answer tuples and of answers based hereupon can be introduced for databases with $V$–relations. Details can be found in [Klein1999].

## 4 Many–valued logics for improving results

If occurrences of $\omega$ are assumed to represent an arbitrary non–empty set of ordinary values (close extension semantics), sure answers of any of the forms defined above are not computable even if only database schemas are considered consisting of a single relation type with three or more attributes. This follows from the undecidability of finite validity in first–order logic (theorem of Trachtenbrot). If occurrences of $\omega$ represent exactly one unknown value (completion semantics), answers are computable. We face, however, a co–NP–complete problem. Hence we have to look for efficient query answering algorithms losing not too much information in general. In the following, we assume completion semantics for partial database relations.

3–valued logic as defined by Łukasiewicz and Kleene (in the logical connectives $\vee$, $\wedge$, and $\neg$ the two logics coincide) can be used for the evaluation of calculus queries against incomplete databases if standard interpretation is adapted appropriately. What has to be avoided, for example, is the identification of independent occurrences of $\omega$ when evaluating expressions like $Rx \wedge Sx$ (e.g.: if $(1,\omega)$ is substituted for $x$ and if $(1,\omega)$ is a tuple in $R$ and $S$ the result of the expression must be *unknown*). As result we get always answers which are subsets of sure uni–answers. This means that the semantical interpretation of single occurrences of $\omega$ is the same for occurrences in the database and in the answer. To get information represented by tuples of sure max–answers into results, the interpretation method has to be changed.

Let $\omega_{\{\}}$ be a new symbol denoting set–valued components of tuples in the following sense: If a partial database relation contains at least two tuples identical up to the values in a single attribute, then this information can be represented by a tuple $p$ having the same values as these two tuples with the exception of the mentioned attribute. In this attribute, $p$ has $\omega_{\{\}}$ as value. In the same manner, tuples with more than one set–valued component can be associated with database relations. The number of additional tuples which have to be considered for each relation is restricted by $min(\binom{n}{2}, k * n)$, with $n$ the size of the relation and $k$ the number of its attributes. With respect to data complexity ([Vardi1986]) no extra complexity is added.

**Example 4:** Let $P = \{(1,2,3),(2,2,4),(1,2,4)\}$. The following additional tuples are associated with $P$: $(1,2,\omega_{\{\}}),(\omega_{\{\}},2,4),(\omega_{\{\}},2,\omega_{\{\}})$.

The knowledge connected with $\omega_{\{\}}$ can be used for the evaluation of expressions where this kind of value is compared to an ordinary value or to an unknown value ($\omega$): We know that in each possible database *pdb* to an incomplete database *idb* there are at least two substitutions for the corresponding variable in the query with different values in $A$. Hence in case of an equality expression $\omega_{\{\}} = c, c \in D$, considered for an interpretation induced by *idb*, there is at least one (ordinary) interpretation induced by *pdb* with value *false* for the corresponding expression. In case of an equality $\omega_{\{\}} = \omega$ or inequality expressions similar conclusions can be drawn.

In our interpretation method the assignment of truth values to comparisons is the same as if we use 3–valued logic and identify occurrences of $\omega_{\{\}}$ and $\omega$ up to the following exceptions:

– comparisons $\omega_{\{\}} = c, c = \omega_{\{\}}, \omega_{\{\}} = \omega, \omega = \omega_{\{\}}, c \in D$, evaluate to *locally false*

– comparisons $\omega_{\{\}} \neq c, c \neq \omega_{\{\}}, \omega_{\{\}} \neq \omega, \omega \neq \omega_{\{\}}, c \in D$, evaluate to *locally true*

under the condition that these expressions do not result from comparisons in the query with the same variable in both operands.

The conclusion that at least two substitutions exist 'consumes' one of these substitutions. This means that the value of an expression has some 'local' quality. If during the evaluation of a formula it 'meets' another value resulting from the same observation, the information it carries must no longer be valid. In order to deal with this kind of locality, we introduce a new 5–valued logic extending 3–valued logic by the values *locally true (LT)* and *locally false (LF). Locally true* as final value of a formula is equated with *true*.

LT/LF-logic:

The connectives $\vee$, $\wedge$, and $\neg$ are defined by the following tables:

| $\vee$ | T | F | LT | LF | U |
|---|---|---|---|---|---|
| T | T | T | T | T | T |
| F | T | F | LT | LF | U |
| LT | T | LT | LT | LT | LT |
| LF | T | LF | LT | U | U |
| U | T | U | LT | U | U |

| $\wedge$ | T | F | LT | LF | U |
|---|---|---|---|---|---|
| T | T | F | LT | LF | U |
| F | F | F | F | F | F |
| LT | LT | F | U | LF | U |
| LF | LF | F | LF | LF | LF |
| U | U | F | U | LF | U |

| $\neg$ | |
|---|---|
| T | F |
| F | T |
| LT | LF |
| LF | LT |
| U | U |

All other connectives are defined by the equivalences of propositional logic. From the tables follows that the rules of associativity, commutativity, and de Morgan are fulfilled. The laws of absorption, however, are not fulfilled.

Because of the finiteness of databases and the range restrictedness of formulas the substitution interpretation of quantifiers is applicable, i.e. quantified subexpressions of calculus queries can be replaced by finite conjunctions and disjunctions. Hence query evaluation using this logic is determined in the standard way by the given tables. There is no loss of efficiency.

It can be shown that the use of LT/LF–logic results in answers which are always supersets of answers obtained by applying 3–valued logic and subsets of the set of sure answer tuples.

**Example 5:** Consider Example 2. With $(1, \omega_{\{\}})$ substituted for $x$ we get:

$(1,\omega_{\{\}}) \in \{(1,2),(1,3)\} \wedge \neg(\omega = \omega_{\{\}}) \equiv T \wedge \neg LF \equiv T \wedge LT \equiv LT$.

The answer relation is the sure max–answer $\{(1,\omega)\}$.

LT/LF–logic does not exploit the information derivable from the independence of substitutions for variables. Consider an equality $x.A = y.B$, for example. If we get $\omega_{\{\}} = \omega_{\{\}}$ after substitution, it is possible to assign the truth value *locally false*. Moreover, $LT \wedge LT$ can be evaluated to $LT$ if the operands result from distinct substitutions.

In order to exploit the independence of substitutions in subformulas, we introduce truth values with sets of query variables as indices. Each set represents possible dependencies of variables. If a comparison expression evaluates to *locally true* or *locally false*, this value gets the set of variables occurring in the corresponding comparison expression of the query as index. The definition of logical connectives has to be changed such that the dependencies are considered correctly (e.g.: $LT_{\{x\}} \wedge LT_{\{x\}} = U, LT_{\{x\}} \wedge LT_{\{y\}} = LT_{\{x,y\}}$).

**Example 6:** Let $R = \{(1,2),(1,3)\}, S = \{(1,\omega)\}$ and $W = \{(1,\omega)\}$. Consider the following query:

$$(x,y)/Rx \;\wedge\; Ry \;\wedge\; \neg(\exists s : Ss)(x.B = s.B \wedge y.B = s.B) \wedge \neg(\exists w : Ww)(x.B = w.B)$$

With substitution $(1,\omega_{\{\}})$ for variables $x$ and $y$ the use of pure LT/LF-logic results in the following computation (in simplified form):

$T \wedge T \wedge \neg(LF \wedge LF) \wedge \neg(LF) \equiv LT \wedge LT \equiv U$

The indexed form gives:

$T \wedge T \wedge \neg(LF_{\{x\}} \wedge LF_{\{y\}}) \wedge \neg(LF_{\{x\}}) \equiv$
$\neg(LF_{\{x\},\{y\}}) \wedge LT_{\{x\}} \equiv LT_{\{x\},\{y\}} \wedge LT_{\{x\}} \equiv LT_{\{y,x\}}$

Changing the evaluation of quantified subexpressions appropriately allows to use LT/LF–logic under close extension semantics, too. Since variables in $V-$relations represent exactly one ordinary value, this logic can be applied also in case of databases with $V$–relations.

## 5 Proof theoretic view compared to model theoretic view

For incomplete databases with variables it is reasonable to use variables in answers. The following definition is from [Lipski1984]: The *closed world assumption response* to $q$ in $idb$ is defined as the set $\{t \mid t\ V\text{–tuple on } \beta \wedge (\forall \nu : \nu \text{ valuation on } V)(\nu(t) \in q(\nu(idb)))\}$. It is argued in [Lipski1984] that the closed world assumption response corresponds to what is simply called a *set of answers* in [Reiter1986] to a *query* $< x_1/\tau_1, ..., x_n/\tau_n \mid \Phi(x_1, ..., x_n) >$ with $\tau_1, ..., \tau_n$ simple types and $\Phi(x_1, ..., x_n)$ an element of the set of well–formed formulas over a *relational language* $R$. From the definitions follows immediately that only those variables can be used in answers which occur in the incomplete database the query is issued against. Thus, for the query in Example 2 with $\omega$ in relation $Q$ replaced by a variable we get an empty answer. In order to represent in a single answer relation the sure information that $(1,2)$ or $(1,3)$ or both tuples are contained in the answer, other representation forms have to be considered.

In [Reiter1986] an efficient but incomplete evaluation algorithm is proposed (for a similar approach see [Vardi1986]). Well–formed formulas of a relational language are transformed into algebraic expressions with incompleteness arising from inclusions which hold instead of equalities in case of disjunctions and existentially quantified subformulas. This loss of information corresponds exactly to the nonconsideration of some tautologies by 3–valued logic. Thus, the following efficient evaluation method for the model theoretic view produces the same results as the proof theoretic algorithm in [Reiter1986]:
Limited interpretation is applied to calculus queries with the following differences to evaluation for total databases:
- $V$–tuples are instances of variables.
- A range term $Rx$ has the value *true* for an interpretation $I$ if $I(x) \in I(R)$; it has the value *false*, if no $t \in I(R)$ exists such that there is some valuation $\nu$ with $\nu(t) = \nu(I(x))$; its value is *unknown*, otherwise.
- Comparisons of the form $\omega_i \theta \omega_i$ have the value *true* if $\theta \in \{\leq, =, \geq\}$ and *false*, otherwise. Every other comparison with variables has the value *unknown*.
- Propositional expressions are evaluated using 3–valued logic.

This evaluation method corresponds to a 'switch'–method for relational algebra expressions where so–called 'sure' and 'maybe'–versions of operations are used depending on the structure of the expression (the operation 'difference' causes a switch from 'sure'–versions of operations to 'maybe'–versions and vice versa). Supposing these evaluation methods to be used we get an equivalence theorem corresponding to the classical one for total databases.

## 6 Conclusions

We have introduced different kinds of answers representing sure information to queries against incomplete databases. A new 5–valued logic has been proposed allowing to improve results of query evaluation without loss of efficiency. We have argued that a suitable use of 3–valued logic in case of missing values represented by variables gives an evaluation method for calculus queries which produces the same results as the proof theoretic algorithm proposed in [Reiter1986].

## References

[Abiteboul *et al.*1991] S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. *Theor. Comp. Sci.*, 78:159–187, 1991.

[Imielinski and Lipski1984] T. Imielinski and W. Lipski. Incomplete information in relational databases. *J. of the ACM*, 31(4):761–791, 1984.

[Klein1999] H.-J. Klein. On the use of marked nulls for the evaluation of queries against incomplete relational databases. In T. Polle, T. Ripke, and K.-D. Schewe, editors, *Fundamentals of Informations Systems*, pages 82–99. Kluwer Academic Publ., 1999.

[Lipski1979] W. Lipski. On semantic issues connected with incomplete information databases. *ACM Trans. on Database Systems*, 4(3):262–296, 1979.

[Lipski1984] W. Lipski. On relational algebra with marked nulls. In *Proc. ACM PODS*, pages 201–203, 1984.

[Maier1983] D. Maier. *The Theory of Relational Databases*. Computer Science Press, 1983.

[Reiter1986] R. Reiter. A sound and sometimes complete query evaluation algorithm for relational databases with null values. *J. of the ACM*, 33(2):349–370, 1986.

[Vardi1986] M. Vardi. Querying logical databases. *J. of Comp. and Syst. Sci.*, 33:142–160, 1986.

[Zimányi and Pirotte1997] E. Zimányi and A. Pirotte. Imperfect information in relational databases. In A. Motro and P. Smets, editors, *Uncertainty Management in Information Systems*, pages 35–87. Kluwer Academic Publ., 1997.