Incomplete Information

Gösta Grahne
Department of Computer Science
Concordia University, Montreal, Canada
grahne@cs.concordia.ca

SYNONYMS

Uncertain information; Null values; Indefinite information

DEFINITION

Incomplete information arises in relational databases, when a fact (tuple) has to be inserted in a relation, and values for some required columns are missing. For instance, in an employee database, the phone number of one employee might be missing, as might also the address of another employee. There are numerous reasons for such missing information, e.g. the insertion was done through a view, or the incomplete tuple originated from another database that does not record these fields. In information integration and data exchange systems incomplete information is rampant. Note that here the null values only represent unknown, existing values. The other main case, namely the one that the column heading is not applicable to the tuple in question, is not covered here, for a treatment, see e.g. [14].

It is easy to *store* missing or incomplete information, by simply using a symbol, say $\perp$, different from the symbols of the domain of the database. As an illustration, consider the following employee database:

Employee	Department	Home Town	Phone no.
Jerry	*Sales*	*New York*	$\perp$
Elaine	*Accounting*	$\perp$	*123*
George	*Services*	*New York*	*456*

In the table above, Jerry's phone number is unknown as is Elaine's home address. The third tuple has complete information. So far the picture is quite uncomplicated. The difficulties arise when *queries* are applied to an incomplete database. For instance, suppose the query asks for all employees living in New York. Should Elaine be included in the answer, as it is not ruled out that she lives in New York? Her unknown hometown could very well be New York. It seems that Elaine *might* be in the answer.

For another example, suppose the relation is decomposed into R(Employee, Department, Home Town) and S(Home Town, Phone no.). Now, joining back R and S, at least the original relation. should be recovered. However, it is not clear how to join the R-tuple (*Elaine, Accounting*, $\perp$) with the S-tuple ($\perp$, *123*). Nevertheless, whatever Elaine's home address is, the null-value in the R-tuple must clearly be the same as the null-value in the S-tuple, as they both represent Elaine's unknown home town. Therefore, the tuple (*Elaine, Accounting*, $\perp$, *123*) should *surely* be in the result of the join.

HISTORICAL BACKGROUND

The enormously successful relational model emerged out of algebra and logic, starting with Codd's fundamental 1970's papers. By the end of the decade, the problem of null values was noted and resulted in some early efforts not covered here, see e.g. [14, 17]. Around this time Ray Reiter also promulgated the idea of "Proof theoretical" vs. "Model Theoretical" interpretation of the relational model (see references in [11]). Roughly, one could say that in proof theory the database is a set of ground facts in First Order Logic, and the query (also a logical formula) answers are something to deduce from the theory by proof-theoretic means. In "Model Theory" on the other hand, the database is a finite structure, and the queries are algebraic operators applied to the databases. Indeed, Reiter proposed a first order interpretation of the null value: Since for a regular tuple, $(a, b) \in r$ is interpreted as a ground fact $R(a, b)$, it is plausible to interpret $(a, x) \in r$, where x is a null value, as the logical sentence $\exists x : R(a, x)$. Reiter then developed query answering algorithms for his (existentially) extended relational theories [11].

At the same time, relational theory started developing, creating its own apparatus, containing lossless joins, tableaux and the chase, in addition to many other useful tools and concepts, such as the "crown jewel" relational algebra. Working along these lines, T. Imielinski and W. Lipski came out in 1981 with a *VLDB* conference abstract of their landmark paper *Incomplete Information in Relational Databases*, published in *J. ACM* in 1984.

First, Imielinski and Lipski call a relation containing null values a *table* (as opposed to a relation). One assumes two disjoint countably infinite sets of *constants*, denoted $a, b, c,$ *Jerry, Elaine*$, \ldots$, and of *variables*, denoted $x, y, z, \ldots$. The table in the introduction, could be written (concisely) as $\{($*Jerry, Sales, New York,* $x), ($*Elaine, Accounting,* $y,$ *123*$), ($*George, Services, New York, 456*$)\}$

Second, [9] adopts the *possible worlds* interpretation of incomplete databases: an *incomplete database is a set of complete databases*, one of which is or corresponds to the real world. In other words, the knowledge about the real world, that the incomplete database has is exactly this set. In the sequel (usually infinite) incomplete databases (sets of complete databases) are denoted by by $\mathscr{X}, \mathscr{Y}, \ldots$.

Tables represent incomplete databases through valuations, that instantiate the null values to ordinary, known values. Formally, a *valuation* is a mapping from the variables to the constants, that is identity on the constants. Valuations are extended to tuples and tables in the obvious homeomorphic way. Now a database r represents one of the possible worlds of T, if there is a valuation v, such that $r = v(T)$. For instance, let $T = \{(a, x), (b, y)\}$. Pretend for a moment that domain of constants is finite and only includes a and b. Then T represents $\mathscr{X}$, where $\mathscr{X} = \{\{(a, b), (b, b)\}, \{(a, a), (b, b)\}, \{(a, b), (b, a)\}, \{(a, a), (b, a)\}\}$. Thus $Rep(T)$, the incomplete database $\mathscr{X}$, represented by a table T, is defined as

(1) $$Rep(T) = \{r : r \supseteq v(T), \text{for some valuation } v\}$$

Note that the *open world* assumption is made, when requiring only $r \supseteq v(T)$. It means that any fact not recorded in all possible databases is considered unknown. Note further that under an open world assumption, a *complete* database r actually represents all supersets of r. Since nothing is surely false, it is clear that querying under the open world assumption cannot include negation. In a *closed world* interpretation, on the other hand, any fact *not* recorded in the database, or not instantiable from a tuple in a table is considered false. For tables T, the closed world interpretation of the possible databases it represents is defined as $Rep_{cwa}(T) = \{r : r = v(T), \text{for some valuation } v\}$. This means that a tuple that does not belong to any $r \in Rep_{cwa}(T)$ is considered false.

The main insight when processing a query q on an incomplete database is as follows: since the "real" database r is one within a set $\mathscr{X}$, the "real" answer should ideally be $q(r)$. However, given only $\mathscr{X}$, what can be captured with query q is the set $q(\mathscr{X}) = \{q(r) : r \in \mathscr{X}\}$. In particular, if $\mathscr{X}$ is represented by a table T, it would be desirable to find a table, denote it $\hat{q}(T)$, such that

(2) $$Rep(\hat{q}(T)) = q(Rep(T)).$$

This is where the technical development can begin.

SCIENTIFIC FUNDAMENTALS

First, note that the open world assumption itself is causing a subtle difficulty when trying to satisfy the commutativity requirement (2). Consider the table T containing a single tuple (a, b). After applying a selection $\sigma_{A=a}$ on T one would (correctly) expect that the tuple (a, b), and nothing else, is in the resulting table $\widehat{\sigma_{A=a}}(T)$. However, equation 2 is not satisfied, since for instance (c, d) occurs in some $r \in Rep(\widehat{\sigma_{A=a}}(T))$, whereas $(c, d) \notin r$, for all $r \in \sigma_{A=a}(Rep(T))$, as such tuples have been dropped by the selection. Unless the closed world assumption is adopted — in which case clearly $Rep_{cwa}(\widehat{\sigma_{A=a}}(T)) = \sigma_{A=a}(Rep_{cwa}(T))$ — condition 2 has to be relaxed.

The key concept for achieving this purpose is the notion of *certain answer*. Let q be a query and $\mathscr{X}$ an incomplete

database. The certain answer to q on $\mathscr{X}$ consists of those tuples that are in $q(r)$, for *every* possible database $r \in \mathscr{X}$, that is, the certain answer is $\cap_{r\in\mathscr{X}} q(r) = \cap(q(\mathscr{X}))$.
Given a fixed query language, two incomplete databases are not distinguishable if they give the same certain answer to every query in the language. Formally, let $\mathcal{Q}$ be a query language (the set of all queries expressible in the language). Then incomplete databases $\mathscr{X}$ and $\mathscr{Y}$ are said to be $\mathcal{Q}$-*equivalent*, if $\cap(q(\mathscr{X})) = \cap(q(\mathscr{Y}))$, for all $q \in \mathcal{Q}$. This $\mathcal{Q}$-equivalence is denoted $\mathscr{X} \equiv_{\mathcal{Q}} \mathscr{Y}$. There is another equivalence relation on incomplete databases called *coinitiality.* Now $\mathscr{X}$ is coinitial with $\mathscr{Y}$, in symbols $\mathscr{X} \approx \mathscr{Y}$, if $\mathscr{X}$ and $\mathscr{Y}$ have the same minimal (w.r.t. subset) elements. Let $\mathcal{Q}_P$ be the set of all queries expressible in *positive* relational algebra (no negation, no inequalities in selection conditions), and let $\mathcal{Q}_{RA}$ be the set of *all* queries expressible in the full relational algebra. It turns out that provably $\mathscr{X} \equiv_{\mathcal{Q}_P} \mathscr{Y}$ iff $\mathscr{X} \approx \mathscr{Y}$, and $\mathscr{X} \equiv_{\mathcal{Q}_{RA}} \mathscr{Y}$ iff $\mathscr{X} = \mathscr{Y}$ [9, 4].
Therefore, if the open world assumption if followed, the best approximation of equation 2 is to weaken it to

$$Rep(\hat{q}(T)) \equiv_{\mathcal{Q}} q(Rep(T)) \tag{3}$$

for some query language $\mathcal{Q}$. This led [9] to the notion of *representation system.* Let T be a class of tables, *Rep* the interpretation function, and $\mathcal{Q}$ a query language. Then the triple $(\mathsf{T}, Rep, \mathcal{Q})$ is a *representation system* if for all $T \in \mathsf{T}$, and all $q \in \mathcal{Q}$, there is a $\hat{q}(T) \in \mathsf{T}$, such that equation 3 is satisfied. Then the certain answer $\cap(q(Rep(T)))$ is clearly equal to $\cap(Rep(\hat{q}(T)))$. This means that the certain answer can be extracted from $\hat{q}(T)$, in some cases efficiently, as will be seen below.
The original paper [9] considers the classes consisting of Codd Tables, Naive Tables, and Conditional Tables. Other classes of tables can be found in [2, 4, 15, 13]. Here, the attention is restricted to the original three table classes.
A *Codd table* is a table where each occurrence of a null value is represented by the same symbol (as for example in the table in the first section of this entry). Let for instance the symbol $\perp$ denote a null value. Evaluating a query q (i.e. computation of $\hat{q}(T)$ in equation 3) proceeds as in regular relational algebra, with the additional evaluation rules saying that $\perp \neq \perp$, and $\perp \neq a$, for all constants a, and as long as the selection conditions are atomic, i.e. of the form $\sigma_{A=a}$ or $\sigma_{A\neq a}$. This is in fact the solution proposed by Codd, and currently implemented in most commercial database management systems. However, the largest query language that Codd tables can support consists of queries expressible in relational algebra using only projection and selection, where, notably, inequalities are allowed in selections. Using any more operators from the relational algebra makes it impossible to satisfy equation 3, that is, for some such queries q there is no Codd table $\hat{q}(T)$ satisfying 3. The same holds under the closed world assumption. In addition to this, selections can have arbitrary Boolean combinations of atomic selection conditions, but then testing whether a tuple satisfies such a condition becomes coNP-complete. This is due to the fact that any propositional formula can be encoded as a compound selection formula. Note however, that the coNP-completeness depends on the fact that the query is part of the input (expression complexity). For data complexity, the evaluation can be carried out in time polynomial in the size of the table.
A *Naive table* is a table using *distinguishable* nulls, denoted by variables $x, y, , \ldots$. Query evaluation uses the rules $x \neq a, x \neq y, x = x$, for all variables x, y and constants a. Thus the tuple (*Elaine, Accounting*, $\perp$) would indeed join with the tuple ($\perp$, *123*), resulting in tuple (*Elaine, Accounting*, $\perp$, *123*). These tuples would of course be represented as (*Elaine, Accounting*, x), and (x, *123*). It turns out that if $\mathcal{Q}_P$ is chosen as query language, then $(\mathsf{T}_N, Rep, \mathcal{Q}_P)$, where T_N is the class of all Naive tables, indeed forms a representation system. It has also been shown that Naive tables can handle recursion, i.o.w. one can add any positive datalog program as a query, and still be able to $\approx$-represent the result [4]. More generally, one can note that query evaluation, i.e. computation of $\hat{q}(T)$, proceeds by treating the null-values as constants, pairwise distinct, and distinct from all the "real" constants. Thus query evaluation in Naive tables has the same computational complexity as in the regular case. A similar result was also independently proved in [16].
Much has been written about certain answers in the database literature, but not always with complete transparency. Recall that the *certain answer* to a query q on $\mathscr{X}$, are those tuples that are in the answer to q for *every* possible database $r \in \mathscr{X}$. In other words, the certain answer to q on $\mathscr{X}$ is $\cap(q(\mathscr{X}))$. Note that $\hat{q}(T)$ is a Naive table that satisfies equation 3, for all positive datalog programs, or algebraic expressions in $\mathcal{Q}_P$. Conveniently, the certain answer can be obtained by retaining all variable-free tuples in $\hat{q}(T)$, as it is easily seen that $\mathscr{X} \approx \mathscr{Y}$ implies $\cap\mathscr{X} = \cap\mathscr{Y}$. The certain answer however looses some information from $\hat{q}(T)$. For a simple example, similar to the one in the introduction, let $T = \{(a, x, c)\}$ with schema (A, B, C). The certain answer to both $\pi_{A,B}(Rep(T))$ and $\pi_{B,C}(Rep(T))$ is empty. On the other hand, the

certain answer to $\pi_{A,C}(\pi_{A,B}(Rep(T)) \bowtie \pi_{B,C}(Rep(T))) = \{(a,c)\}$. Here $\widehat{\pi_{A,B}}(T) = \{(a,x)\}$, $\widehat{\pi_{B,C}}(T) = \{(x,c)\}$ and $\{(a,x)\widehat{\bowtie}\{(x,c)\} = \{(a,c)\}$. Note how variables can be shared over several tables, above the x the the left-hand side of the join is the same as the x on the right-hand side of the join. (On the other hand, for instance $\{(a,x)\widehat{\bowtie}\{(y,c)\} = \emptyset$.) Thus, if the query answer is to be used as a *view* for further querying, the answer should consist of all of $\hat{q}(T)$, not just the variable-free tuples. This aspect has been emphasized in [5]. Furthermore, as Lipski has shown in an all but forgotten paper [10], if query evaluation is to be *uniformly recursive* (such as the one for Naive tables), all tuples in $\hat{q}(T)$ need to be stored in the view.
Conditional tables. Naive tables form a representation system for the positive fragment of relational algebra (and for positive recursion). However, allowing set difference or inequalities in selection conditions in the queries might produce results not expressible as Naive tables. Consider for instance $\sigma_{A\neq a}(Rep(T))$, where $T = \{(a,b),(c,d),(x,e)\}$. A moments reflection will reveal the fact that $\sigma_{A\neq a}(Rep(T)) = \{\{(c,d)\},\{(c,d),(a,e)\},\{(c,d),(b,e)\},\{(c,d),(d,e)\},\ldots\}$. It is easy to see that there is no Naive table that can represent this set. The problem is that the minimal elements in this set either has one tuple (when $x = a$), or two tuples (when $x \neq a$). The minimal elements in $Rep(T)$, for any (non-redundant) Naive table T, each have the same number of tuples. This problem can be overcome by using a stronger class of tables, albeit on the expense of tractability of query evaluation. This stronger class is called Conditional tables, here denoted $\mathsf{T_C}$. A Conditional table is like a Naive Table, with the addition that each tuple has an associated condition, formed by a Boolean combination of atoms of the form $x = y, x \neq y, x = a, x \neq a, a = b, a \neq b$, for constants a, b and variables x, y. Note that for two distinct constants a and b, condition $a = b$ is tautologically false, and condition $a \neq b$ is tautologically true. In our previous example, $\sigma_{A\neq a}(Rep(T))$ can be represented by the Conditional table $U = \{(c,d); a = a, (x,e); x \neq a)\}$. The incomplete database represented by a conditional table T is defined as

$$(4) \qquad Rep(T) = \{r \supseteq v(T) : v \text{ is a valuation}, v(T) = \{v(t) : t \in T \text{ and } v \text{ makes the condition in } t \textit{ true}\}\}$$

For example, if v is the valuation $x \mapsto a$, and v' the valuation $x \mapsto b$, then, for U as above, $v(U) = \{(c,d)\}$ as the condition of the second tuple $(x,d); x \neq a$ becomes false. On the other hand, $v'(U) = \{(c,d),(b,e)\}$, since $v'(x \neq a)$ becomes $b \neq a$, which is true. Set difference is treated in a similar manner, for instance, subtracting $\{(c,b),(e,b)\}$ from $\{(x,b)\}$ results in the conditional table $\{(x,b); x \neq c \wedge x \neq e\}$. More details are given in (crossref). All in all, it holds that $(\mathsf{T_C}, \mathsf{\mathit{R}ep}, \mathcal{Q}_{\mathsf{P}\neq})$, is a representation system. Here $\mathcal{Q}_{P\neq}$ denotes the query language obtained from $\mathcal{Q}_P$, by allowing inequalities in selection conditions. If set difference is to be incorporated, the closed world assumption has to be adopted. In this case it holds that $(\mathsf{T}_C, Rep_{cwa}, \mathcal{Q}_{RA})$, is a representation system, actually satisfying the stronger condition 2. Positive datalog recursion can also be added both under the open and the closed world assumption.
For the computational complexity of query evaluation, recall that query evaluation in the system $(\mathsf{T}_N, Rep, \mathcal{Q}_P)$ is polynomial in the number of tuples in T. Conditional tables are more complex. For all $q \in \mathcal{Q}_{RA}$ and conditional tables T, the conditional table $\hat{q}(T)$ can be computed in polynomial time. However, the conditions in table $\hat{q}(T)$ can have a convoluted structure. Thus testing whether a tuple t is in the certain answer, i.e. if $t \in \cap(Rep(\hat{q}(T)))$ is a coNP-complete problem. This result holds even if T is a Codd table, in which case a query $q \in \mathcal{Q}_{RA}$ is needed to get the lower bound (here the fact that $\hat{q}(T)$ is not a Codd table is ignored). On the other hand, if taking an arbitrary table $T \in \mathsf{T_C}$, the query can be identity. In other words, the set $\cap(Rep(T))$ has a coNP-complete membership test. These, and further complexity results can be found in [2].
The theory of representation systems has also been extended to incorporate dependencies, see [9, 4]. Let $\mathscr{X}$ be an incomplete database, and Σ a set of equality and weakly acyclic tuple generating dependencies. Denote by $\Sigma(\mathscr{X})$ the set of all minimal relations s, such that $s \supseteq r$, for some $r \in \mathscr{X}$, and s satisfies all dependencies in Σ. It has been shown that for all Naive tables T, there is a Naive table $\Sigma(T)$, such that $Rep(\Sigma(T)) \approx \Sigma(Rep(T))$. If conditional tables are used, the coinitiality can be replaced by equality.
The relationship between tables and constraint databases is explored in [12]. The relationship between tables and probabilistic databases, and between tables and data provenance is elegantly captured in the semiring framework of [7].

KEY APPLICATIONS

Three important applications of incomplete information will be discussed here. The first one is the problem of *view updates* (crossref). In this scenario, a view is defined by a query q, but the view is virtual, and only the

database is materialized. As queries almost never are one-to-one functions, when a tuple t is inserted through a virtual view $q(r)$, it has to be translated to an insertion of a tuple, say $\hat{t}$, into r, s.t. $q(r \cup \{\hat{t}\}) = q(r) \cup \{t\}$. There is also a further requirement, that essentially guarantees the "minimality" of the change caused be the insertion of $\hat{t}$. Since there in general are several, sometimes infinitely many, insertions $\hat{t}$ that qualify, there actually is a set of possible databases $\{r \cup \{\hat{t}\} : \hat{t} \text{ qualifies }\}$. For example, for $q = R \bowtie S$, where $R(A, B)$ and $S(B, C)$, the deletion of a tuple (a, b, c) from a view $q(r, s)$, can be accomplished by either deleting the tuple (a, b) from r, or deleting the tuple (b, c) from s. This can easily be achieved if the database is stored as a conditional table, by simply replacing the tuples (a, b) in r, and (b, c) in s, with the conditional tuples $(a, b); x = 1$, and $(b, c); x \neq 1$, where x is an arbitrary fresh variable.
The views in the view update scenario are virtual. If views are materialized, they can be used to speed up query processing. The case where all views are materialized, and the database virtual is the classic information integration scenario (crossref). The views represent data sources, and the database schema represents the integrated schema that queries are formulated over. Intuitively, one can imagine that the integrated database exists, the view-defining queries are executed, and the views are accordingly populated. After this, the integrated database disappears. It is easy to see that a set of materialized views represent a set of possible databases, each satisfying the requirement that the current content of the views can be derived from it. Note that there is an open world assumption, as the requirement is that the view tuples are a subset (not necessarily proper) of $q(r)$, for all view definitions q and possible databases r. The closed world assumption would require equality, not just subset. For query answers, the certain answer is usually desired. The certain answer means in this context the set of tuples that are in the query result, for every possible database. To answer queries over the integrated schema, one can either rewrite the query in terms of the view-schemas, or reconstruct a representation of all the possible databases, and evaluate the original query on this representation. It perhaps speaks for the robustness of the concept of conditional tables, that they can do the job of representing the set of possible databases, whenever the views are queries in $\mathcal{Q}_{P\neq}$, or in $\mathcal{Q}_{RA}$ if the closed world assumption is adopted [1, 6].
The last application is that of *data exchange* (crossref). The setting is a peer-to-peer system, where each peer has a relational database and wants to exchange tuples with other peers. The schemas of any two peers, say p_1 and p_2 are usually different from each other, so the data has to be converted when exchanged. This is achieved by defining a mapping from p_1 to p_2. This mapping could for instance be expressed as an equation $q_1(p_1) \subseteq q_2(p_2)$. When we export data from p_1 to p_2, we evaluate q_1 on p_1, and make sure that the resulting tuples are in $q_2(p_2)$. Since the tuples have to be inserted into p_2, it is easy to see that this is similar to the familiar problem of view updates, and thereby also incomplete information. For a very simple example, suppose the schema of p_1 is $R(A, B)$, and the schema of p_2 is $S(A, C)$. Consider then the equation $\pi_A(R) \subseteq \pi_A(S)$. If r contains a tuple (a, b), it is clear, that in order to satisfy the equation, a tuple (a, x) has to be in s. It has for example been shown [8], that if $q_1 \in \mathcal{Q}_P$ and q_2 only uses projection, then given null-free instances of p_1 and p_2, there is a naive table $p_2\prime$, containing p_2, representing all minimal solutions to the equation, that is $q_1(p_1) \subseteq q_2(r)$, for all $r \in Rep(p_2')$, and the equation is not satisfied by any proper subsets of these r's.

FUTURE DIRECTIONS

It is clear that new applications, new data models, and new query languages will encounter the problem of incomplete information. In order not to "re-invent the wheel," any solution should build on the foundation laid in [9]. A step in this direction has for instance been taken in [3].

RECOMMENDED READING

[1] Serge Abiteboul, Oliver M. Duschka: Complexity of Answering Queries Using Materialized Views. *PODS* 1998: 254-263

[2] Serge Abiteboul, Paris C. Kanellakis, Gösta Grahne: On the Representation and Querying of Sets of Possible Worlds. *Theor. Comput. Sci.* 78(1): 158-187 (1991)

[3] Serge Abiteboul, Luc Segoufin, Victor Vianu: Representing and querying XML with incomplete information. *ACM Trans. Database Syst.* 31(1): 208-254 (2006)

[4] Gösta Grahne: *The Problem of Incomplete Information in Relational Databases.* Springer 1991

[5] Gösta Grahne, Victoria Kiricenko: Towards an algebraic theory of information integration. *Inf. Comput.* 194(2): 79-100 (2004)

[6] Gösta Grahne, Alberto O. Mendelzon: Tableau Techniques for Querying Information Sources through Global Schemas. *ICDT* 1999: 332-347

[7] Todd J. Green, Gregory Karvounarakis, Val Tannen: Provenance semirings. *PODS* 2007: 31-40

[8] Leonid Libkin: Data exchange and incomplete information. *PODS* 2006: 60-69

[9] Tomasz Imielinski, Witold Lipski Jr.: Incomplete Information in Relational Databases. *J. ACM* 31(4): 761-791 (1984)

[10] Witold Lipski Jr.: On Relational Algebra with Marked Nulls. *PODS* 1984: 201-203

[11] Raymond Reiter: A sound and sometimes complete query evaluation algorithm for relational databases with null values. *J. ACM* 33(2): 349-370 (1986)

[12] Peter Z. Revesz: *Introduction to Constraint Databases.* Springer 2002

[13] Anish Das Sarma, Omar Benjelloun, Alon Y. Halevy, Jennifer Widom: Working Models for Uncertain Data. *ICDE* 2006: 7

[14] Carlo Zaniolo: Database Relations with Null Values. *PODS* 1982: 27-33

[15] Tomasz Imielinski, Kumar V. Vadaparty: Complexity of Query Processing in Databases with OR-Objects. *PODS* 1989: 51-65

[16] Moshe Y. Vardi: Querying Logical Databases. *J. Comput. Syst. Sci.* 33(2): 142-160 (1986)

[17] Yannis Vassiliou: Null Values in Data Base Management: A Denotational Semantics Approach. *SIGMOD Conference* 1979: 162-169