Incomplete Databases: Missing Records and Missing Values

Werner Nutt, Simon Razniewski, and Gil Vegliach

Free University of Bozen-Bolzano, Dominikanerplatz 3, 39100 Bozen, Italy
{nutt, razniewski}@inf.unibz.it, gil.vegliach@gmail.com

Abstract. Data completeness is an essential aspect of data quality as in many scenarios it is crucial to guarantee the completeness of query answers. Data might be incomplete in two ways: records may be missing as a whole, or attribute values of a record may be absent, indicated by a null.
We extend previous work by two of the authors [10] that dealt only with the first aspect, to cover both missing records and missing attribute values. To this end, we refine the formalization of incomplete databases and identify as an important special case the one where values of key attributes are always known. We show that in the presence of nulls, completeness of queries can be defined in several ways. We also generalize a previous approach to specifying completeness of parts of a database, using so-called table completeness statements. With this formalization in place, we define the main inferences for completeness reasoning over incomplete databases and present first results.

1 Introduction

Data quality deals with the question how well data serves its purpose. Aspects of data quality concern accuracy, currency, correctness, and similar issues. In settings such as manual data insertion or data integration, completeness of data plays a key role. The issue is to decide whether data is complete enough for its aims and what might be potentially missing. In particular, in relational databases incompleteness comes in two flavors: records (rows) in tables might be missing or attribute values might be *null*.

Consider as a driving example the management of school data in the province of Bolzano, Italy, which motivated the technical work reported here. The schools in the province are largely autonomous in their administration: although the provincial IT department runs a central database for administering data about pupils, teachers and the like, the schools are to a large degree free to choose to which extent to use this system. This freedom leads to many kinds of incompleteness of the underlying database, especially when data submission is optional: for example, when statistics about schools are computed, missing records and *null* values of attributes might have two meanings, either data about facts have not been submitted, or the facts do not hold in reality.

The IT department already has some information about how the schools use the central database, but not yet a systematic approach to apply it. They would like to have a generic technique to tell whether their database is complete enough to answer a certain query, and furthermore what data is needed for the query to be answered completely.

We believe similar problems also occur in other application domains and identify the following research questions:

1. How can one describe completeness of parts of a possibly incomplete database?
2. How can one characterize the completeness of query answers?
3. How can one infer completeness of query answers from such completeness descriptions?

There has been some previous work by two of the authors [**?**,10] on these questions, which only considered incompleteness in the form of missing records. In practice however, incompleteness in the form of *null* values is at least as important.

A problem with SQL *null*s is their ambiguity, as they may not only mean that an attribute value exists, but is unkown, but also that no value applies to that attribute. The established models of null values in database research, such as Codd, v-, and c-tables [6], usually avoid this ambiguity by concentrating on the aspect of unknown values. In this work, we consider the ambiguous standard SQL *null* values [2], because those are the ones commonly used in practice. We will show later that some ambiguity can be resolved, when meta information about database completeness is present.

In this paper, we define a formal framework to study problems that arise when reasoning about the completeness of query answers over databases with *null* values and potentially missing records. We proceed as follows: in section 2 we introduce two example scenarios, in section 3 we formalize incomplete databases, query completeness, and table completeness, in section 4 we show how canonical table completeness statements link table and query completeness, in section 5 we introduce the reasoning problems and we present preliminary results for some of them, in section 6 we discuss related work and with section 6 we conclude this paper.

2 Example Scenarios

We define two scenarios that display incompleteness in the form of missing records and of *null* values. The scenarios differ in that in the first the values of the key attributes are known while in the second some key values are unknown. As we will see later on, in the first case completeness of queries can be detected more easily than in the second.

A School Database. We consider a school database that contains, inter alia, the following two tables, where key attributes are underlined:

- `student` (`sid`, `name`, `level`, `code`, `hometown`)
- `class` (`level`, `code`, `formTeacher`, `viceFormTeacher`, `profile`).

The `student` table stores for each student their unique student ID, their name, the level and the code of the student's class, such as '3' 'A', and the student's hometown. The `class` table stores for each class its level and code, the ID of its form teacher and its vice form teacher, and the profile of the class, such as 'science' or 'commerce'. The attributes `level` and `code` uniquely determine a class while the student ID uniquely determines a student.

Student tuples could be missing because students enroll by submitting a paper form, while the data are only later entered into the electronic database. Similarly, the formation of classes is decided during an administrative meeting, yet this information is not always immediately recorded in the database.

Values for the attributes `level` and `code` of the `student` table may be *null* because (i) students are assigned to classes only some time after their enrollment, and (ii) the decision may not be recorded immediately. For similar reasons, the value of the attributes `formTeacher`, `viceFormTeacher`, or `profile` of the `class` table could be *null*.

In contrast, the keys of tuples can never be *null*. To insert a student record into the database, it is necessary to assign a student ID. Similarly, to insert a class it is necessary to specify its level and code.

Over this school database, some queries will return the same answer over an instance with *null*s and where records are missing as they would over an ideal complete instance. For instance, if all classes have been entered, we can tell the number of classes, even if the form teachers have not yet been entered. Similarly, if each type of profile has been entered once, we know the complete spectrum of profiles, even if the `profile` of some specific class record is *null* or if the class record is missing altogether.

We will sketch later on how to formalize such facts and how to reason about them.

An Integrated Business Database. We consider a company that wants to integrate data about business contacts from different departments into one database. We assume that so far the sales, purchase and research department maintained their own databases with different schemas and custom IDs.

In the integrated database, contacts are identified by their name, address, and city. Even if the original databases were complete, records can be missing because a business contact is kept by another department than those three and values can be missing because the integrated schema contains attributes not present in one of the original databases. Also key values can be missing, because information that makes up the key was missing in one of the original databases.

3 Formalization

3.1 Standard Definitions

In the following we summarize the standard formalization of relational databases and conjunctive queries (cf.[1]). The latter model the widely-used single-block SQL queries. We extend this formalization to take into account SQL-style null values and the semantics of queries over databases with nulls, following the approach in [5].

We assume a set of relation symbols Σ, the *schema*, and an infinite set of constants *dom*, including the rational numbers. A *term* is a constant or a variable.

A *relational atom* is an expression $R(\bar{t})$, where R is a relation symbol and $\bar{t}$ is a vector of terms. A *comparison* is an atom with one of the predicates $<$, $\leq$, $=$, or $\neq$. A *condition* G is a set of relational and comparison atoms. We write a condition as a sequence of atoms, separated by commas. A condition is *safe* if each of its variables occurs in a relational atom. We generically use the symbol L for the subcondition of G containing the relational atoms and M for the subcondition containing the comparisons. A *conjunctive query* is written in the form $Q(\bar{s}) :- B$, where B is a safe condition, $\bar{s}$ is a vector of terms, and every variable in $\bar{s}$ occurs in B. As usual, we call $Q(\bar{s})$ the *head* and B the *body* of Q. A variable in B is called a *join variable* if it occurs at least twice in B. We often refer to the entire query by the symbol Q.

A *database instance* D is a finite set of relational ground atoms, that is, atoms without variables, which may contain the special constant *null*. For a relation symbol $R \in \Sigma$ we write $R(D)$ to denote the set of R-records in D, that is, the set of atoms in D with relation symbol R.

An *assignment* α for the query Q is a mapping that maps the variables of Q to elements of $dom \cup \{null\}$. If A is an atom, then αA is the (ground) atom obtained by replacing every variable x in A with the constant $\alpha(x)$. We say that α *satisfies* the condition $G = L, M$ over the instance D if (i) α maps all join variables of G to constants $\neq$ *null*, (ii) $\alpha A \in D$ for every relational atom $A \in L$, and (iii) α satisfies all comparisons in M. Note that part (i) captures the semantics of *null*s in SQL, where an equality or comparison involving *null* has the truth value "unknown" and thus does not contribute to a query result.

In SQL, a query $Q(\bar{s}) :- B$ can be evaluated under two semantics. Under *bag semantics*, which is the default, applying Q to D results in the multiset $Q^b(D) = \{\!|\, \alpha(\bar{s}) \mid \alpha \text{ satisfies } B \,|\!\}$, which contains as many tuples, including duplicates, as there are satisfying assignments for the body of Q. Under *set semantics*, which is enforced by adding the keyword DISTINCT, Q returns $Q^s(D)$, the set version of $Q^b(D)$, obtained by dropping duplicates.

As usual, we say that a query Q is *set contained* (*bag contained*) in a query Q', if $Q^s(D) \subseteq Q'^s(D)$ ($Q^b(D) \subseteq Q'^b(D)$) for all database instances D. For both semantics, there is a rich body of literature on algorithms and complexity of containment for conjunctive queries (cd. [1]). The problem has also been studied over databases with SQL-style null values [5].

3.2 Incomplete Databases

A database can be incomplete only with respect to a comparison database, considered to be complete. Consequently, we model an incomplete database in the style of Levy [7] as a pair $\mathcal{D} = (D^i, D^a)$ of database instances: D^i, the *ideal* state, with complete information, and D^a, the *available* state, with possibly incomplete information. We require that D^a contains no more information than D^i and formalize this by the concept of dominance. An atom $R(\bar{s})$ is *dominated* by an atom $R(\bar{t})$, written $R(\bar{s}) \preceq R(\bar{t})$, if $R(\bar{s})$ is the same as $R(\bar{t})$, except that it may have more *null*s. Now, for $\mathcal{D}$ to be an incomplete database, the instance D^a must be *dominated* by the the instance D^i, written $D^a \preceq D^i$, in the sense that every atom in D^a is dominated by some atom in D^i.

We say that $\mathcal{D} = (D^i, D^a)$ satisfies the principle of *unique dominance* when the dominance can be established without using any tuple in the ideal database twice, that is, if $R(\bar{s}_1) \preceq R(\bar{t})$ and $R(\bar{s}_2) \preceq R(\bar{t})$ implies $R(\bar{s}_1) = R(\bar{s}_2)$ for all $R(\bar{s}_1), R(\bar{s}_2) \in D^a$ and $R(\bar{t}) \in D^i$. For instance, unique dominance is satisfied by an incomplete database where a key is defined for each relation and no key attribute is *null*. Then every record in the available database is dominated by the record with the same unique identifier in the ideal database.

In data integration, however, key values may uniquely identify records in each of the source databases, but may fail to identify the entities in the integrated database. In such a case, two records from two distinct sources may represent the same entity, but may erroneously be mapped to two records in the integrated database, which are

then dominated by a single record in the ideal database. In such a scenario, unique dominance does not hold.

Example 1. Table 1 shows an incomplete database in the Bolzano school scenario. Information present in the ideal but missing in the available database is written in *italics*. For example, the student Diego is missing entirely and for class 1B we do not know the form teacher and the vice form teacher. Note that we also have *null* values in the ideal database, which express, first, that class 2A has no vice form teacher and, second, that Andrea is registered as an external student not belonging to any class.

D^i		D^a	
`class`	`student`	`class`	`student`
(1, A, 101, *103*, science)	(702, Paul, 1, A, *Bolzano*)	(1, A, 101, *null*, science)	(702, Paul, 1, A, *null*)
(1, B, *104*, *109*, commerce)	(781, Maria, 1, A, *Merano*)	(1, B, *null*, *null*, commerce)	(781, Maria, 1, B, *null*)
(2, A, *102*, *null*, *science*)	(739, Andrea, *null*, *null*, Brunico)	(2, A, *null*, *null*, *null*)	(739, Andrea, *null*, *null*, Brunico)
	(754, Diego, 2, A, Bolzano)		—

Table 1. An incomplete school database

In the school example, unique dominance did hold. We now give an example from the business scenario where unique dominance is not satisfied.

Example 2. The sales, purchase, and research department of a company maintain databases with business contacts, each with a different schema, as shown in Table 2.

Sales	Purchase	Research
`customer(name, street, city)`	`supplier(name, city)`	`partner(name, city, long_term)`
(Johnson Corp., North St., Boston)	(Smith Inc., Detroit)	(Johnson Corp., Boston, no)

Table 2. Contact databases of the three departments

In addition to Johnson Corp. and Smith Inc., the company is also in contact with Miller & Co., through its human resources department. Thus, each of the three should appear once in the integrated database, shown in Table 3. However, due to a failure in entity resolution, Johnson Corp. shows up twice, while Miller Inc. is missing completely. Moreover, some attribute values are *null*, as the corresponding information was missing in the original sources (information missing in D^a is in *italics*).

D^i	D^a
`contact(name, street, city, long_term)`	`contact(name, street, city, long_term)`
(Johnson Corp., North St., Boston, *no*)	(Johnson Corp., North St., Boston, *null*)
	(Johnson Corp., *null*, Boston, no)
(Smith Inc., *Main St.*, Detroit, *yes*)	(Smith Inc., *null*, Detroit, *null*)
(Miller & Co., Central Rd., New York, no)	—

Table 3. Incomplete integrated contact database

Observe that Johnson Corp. appears twice in the available database, coming from the sales department and the research department, but only once in the ideal database. Hence, unique dominance is not satisfied in this example.

3.3 Query Completeness

We now want to define formally when a query Q is complete over an incomplete database $\mathcal{D} = (D^i, D^a)$. Considering bag and set semantics and the concept of dominance, there are three meaningful definitions:

Bag Completeness: The query returns the same *bag* of answers over the available and over the ideal database, that is, $Q^b(D^a) = Q^b(D^i)$;

Set Completeness: The query returns the same *set* of answers over the available and over the ideal database, that is, $Q^s(D^a) = Q^s(D^i)$;

Set Completeness Modulo Redundancy: The query returns all the answers from the ideal database also over the available one, and every (additional) tuple over the available database is dominated by some tuple over the ideal database, thus being redundant, that is, $Q^s(D^i) \subseteq Q^s(D^a)$ and $Q^s(D^a) \preceq Q^s(D^i)$.

If for a query Q, an incomplete database $\mathcal{D}$ satisfies query completeness in one of those cases, we write $\mathcal{D} \models \mathit{Compl}(Q)$. When the meaning is not clear from the context, we add $\cdot^s$, $\cdot^b$, or $\cdot^{sred}$ as superscript to the statement. We call these expressions *query completeness* statements or for short *QC* statements.

Example 3. In the school scenario, consider the query $Q_{lev1}() :- \mathit{student}(s, n, 1, c, h)$. Note that under bag semantics, Q_{lev1} returns a copy of the empty tuple for each student at class level 1 and thus reports the number of such students. Since Q_{lev1} returns the empty tuple twice over both the ideal and the available database, it is bag complete.

Consider also the query $Q_{sci}(n) :- \mathit{student}(s, n, l, c, h), \mathit{class}(l, c, fT, vFT, \text{science})$, which asks for the names of all students in science classes. As 'Diego' is returned over the ideal, but not over the available database, Q_{sci} is neither bag nor set complete.

Clearly, Compl^b(Q) entails Compl^s(Q). Also, Compl^s(Q) entails Compl^{sred}(Q). However, in general, the converse does not hold as we show next.

Example 4. In the business scenario, the query $Q_{Bo}(n, lt) :- \mathit{contact}(n, s, \text{Boston}, lt)$ asks for the name and long term status of contacts from Boston. Over the ideal database it returns $Q^s_{Bo}(D^i) = \{ (\text{Johnson Corp.}, \text{no}) \}$, while over the available database it returns $Q^s_{Bo}(D^a) = \{ (\text{Johnson Corp.}, \text{no}), (\text{Johnson Corp.}, \mathit{null}) \}$. Thus, Q_{Bo} is not set complete over this incomplete database. However, as the additionally returned tuple is dominated by the record returned over the ideal database, Q_{Bo} is set complete modulo redundancy.

3.4 Table Completeness

In addition to the completeness of queries, which can be expressed by QC statements, we want to state the completenes of parts of an incomplete database. To this end, we generalize the table completeness (TC) statements introduced in [10]. A TC statement says that a specific fragment of a relation is complete without requiring other parts of the database to be complete. The challenge is to come up with a formalization that is both intuitive and allows one to infer query completeness from table completeness.

A TC statement, written $\mathit{Compl}(R(\bar{s}); P; G)$, has three components: (i) a relational atom $R(\bar{s})$, (ii) a set of numbers $P \subseteq \{ 1, \ldots, \text{arity}(R) \}$, and (iii) a condition G such that $R(\bar{s}), G$ is safe. The numbers in P are interpreted as attribute positions of R. For instance,

if R is the table `student`, then $\{2, 5\}$ would refer to the attributes `name` and `hometown`. Intuitively, such a TC statement says that if we take the ideal records $R(\bar{t}) \in R(D^i)$, satisfying the conditions $\bar{t} = \bar{s}$ and G over D^i, and project these records onto P, then these projections are also present in $\pi_P(R(D^a))$, the projection of R onto P over the available database. Clearly, we obtain two different semantics, depending on whether the projection returns a bag or a set of records.

Example 5. Taking into account that `profile` is the 5th attribute of the table `class` in the school database, the TC statement $C_{prf} = Compl(\texttt{class}(l, c, fT, vFT, p); \{5\}; true)$ says, under set semantics, that all class profiles are present in the available database. In our example, C_{prf} holds as both 'science' and 'commerce' are in the available database. The TC statement $C_{outBZ} = Compl(\texttt{student}(id, n, l, c, h); \{\}; h \neq \text{Bolzano})$ has an empty set of positions and thus talks about empty tuples. Interpreted under bag semantics, it says that there are as many students from outside Bolzano in the available database as there are in the ideal database. In our example, C_{outBZ} does not hold under bag semantics, as two such students can be found in the ideal, but only one in the available database. It holds under set semantics, though, as both ideal and available database contain at least one student from outside Bolzano.

To make this formal, we associate to the TC statement $C = Compl(R(\bar{s}); P; G)$ the query $Q_C(\bar{s}) :- R(\bar{s}), G$, which returns the R-records $\bar{t}$ satisfying $\bar{t} = \bar{s}$ and G. This query will be evaluated under set semantics over D^i, as we are only interested in the R-records as such, not how many times they can be derived using G. Recall that evaluation of Q_C under set semantics is indicated by the superscript $\cdot^s$. Similarly, we use the operators π^b_P and π^s_P to distinguish between projection on P under bag and set semantics. We now define that $\mathcal{D} = (D^i, D^a)$ *satisfies* C under bag or set semantics, respectively, if

$$\pi^\star_P(Q^s_C(D^i)) \subseteq \pi^\star_P(R(D^a)),$$

where $\star \in \{b, s\}$. The inclusion $\subseteq$ is bag inclusion if $\star = b$ and set inclusion if $\star = s$. To indicate whether a TC statement is to be interpreted under bag or set semantics, we write $Compl_b$ and $Compl_s$, when necessary. In the special case that P comprises all attributes of R, which was the case studied in [10], we can drop the projection and need not distinguish between bag and set semantics, as C is satisfied by $\mathcal{D}$ if $Q^s_C(D^i) \subseteq R(D^a)$.

The next example shows that table completeness statements can also resolve the inherent ambiguity of *null* values found in an available database.

Example 6. Consider the record `student`(739, Andrea, *null*, *null*, Brunico) in our available school database, with *null* values for class level and code. Without further information, we do not know whether level and code are missing or whether the student is an external student not assigned to any class. If we knew, however, that our partial database satisfied $Compl(\texttt{student}(id, n, l, c, h); \{1, 2, 3, 4, 5\}; true)$, that is, the `student` table is complete, we could conclude that the *null*s can only have the meaning that no level and code apply to the student, and hence he is an external.

4 Canonical Table Completeness Statements

The overall goal of reasoning about completeness is to infer QC statements from information about the content of a database, expressed by TC statements.

In previous work on completeness reasoning for databases without *null*s [**?**], a powerful approach consisted in translating a completeness statement about a query Q into a set C_Q of so-called *canonical* TC statements for Q that, intuitively, express which parts of which tables should be complete to guarantee completeness of Q.

Canonical TC statements were then used to reduce the problem of deciding whether an arbitrary set of TC statements C entails the QC statement $Compl(Q)$ (called *TC-QC reasoning*) to checking whether C entails the canonical TC statements C_Q, which is a special case of deciding entailment between sets of TC statements (called *TC-TC reasoning*). TC-TC reasoning was then reduced to the well-studied query containment.

In this section we report on approach to generalize this work to the richer setting accommodating *null*s.

4.1 Definition of Canonical TC Statements

We first want to single out those attributes of relations that must be complete in the available database so that we can answer a query Q completely. These should be the attributes that occur in selections, in joins, and that are output by Q.

Let $Q :- A_1, \ldots, A_n, M$ be a query with relational atoms A_j and a set of comparisons M. A term t occurring in Q is *essential* if (i) t is a constant or (ii) t is a variable occurring more than once in Q. Intuitively, essential terms are those that express a selection condition, a join condition, or that appear in the head of Q. A position p in the relational atom A_i is *essential* if the term occurring at position p in A_i is essential in Q. The set of essential positions of A_i in Q is denoted as $EPos(A_i, Q)$.

The *canonical completeness statement* C_{A_i} for A_i has the form

$$Compl(A_i;\ EPos(A_i, Q);\ A_1, \ldots, A_{i-1}, A_{i+1}, \ldots, A_n, M).$$

Intuitively, a canonical statement $C_{R(\bar{s})}$ states that the projection on the essential positions of $R(\bar{s})$ is complete for those tuples $\bar{t}$ in R that satisfy $\bar{t} = \bar{s}$ and the condition composed by all the other atoms in the query. The *set of all canonical completeness statements* for a query Q is denoted as C_Q. As other TC statements, a canonical statement can be interpreted with respect to set and bag projection, which is indicated by the superscripts $\cdot^s$ and $\cdot^b$ as in $C^s_{A_i}$ and C^b_Q.

4.2 Properties of Canonical TC Statements

Canonical statements are a link between QC and TC statements.

Theorem 1. *Let Q be a conjunctive query, $\mathcal{D}$ an incomplete database. Then we have:*

1. *If $\mathcal{D} \models C^s_Q$, then $\mathcal{D} \models Compl^{sred}(Q)$;*
2. *If $\mathcal{D}$ satisfies unique dominance, then $\mathcal{D} \models C^b_Q$ if and only if $\mathcal{D} \models Compl^b(Q)$.*

The theorem says that canonical statements under set semantics are a sufficient condition for set completeness modulo redundancy. Moreover, in the presence of unique dominance, canonical statements under bag semantics completely characterize bag completeness of a query. The proof is omitted due to space constraints.

As a corollary we note that we can find sufficient conditions for TC-QC entailment in terms of TC-TC entailment. We do not know whether the converse holds, too.

Corollary 1. *Let C be a set of TC statements and Q be a conjunctive query. Then*

1. $C^s \models C^s_Q \;\Rightarrow\; C^s \models \mathit{Compl}^{sred}(Q)$;
2. $C^b \models C^b_Q \;\Leftrightarrow\; C^b \models \mathit{Compl}^{b}(Q)$.

The next corollary is a trivial consequence of Theorem 1.1 and is stated explicitly as a contrast to Theorem 2 below.

Corollary 2. *Let $\mathcal{D}$ be a partial database that satisfies all possible TC-statements and let Q be a conjunctive query. Then $\mathcal{D} \models \mathit{Compl}^{sred}(Q)$.*

We next show that the assumption about unique dominance above cannot be dropped.

Theorem 2. *There exist an incomplete database $\mathcal{D}$ without unique dominance and a conjunctive query Q such that*

1. *$\mathcal{D}$ is complete for all possible TC statements, both under bag and set semantics;*
2. *Q is neither bag nor set complete over $\mathcal{D}$.*

Proof. Let $\mathcal{D}$ consist of $D^i = \{ R(a, b) \}$ and $D^a = \{ R(a, b), R(a, null) \}$. Unique dominance does not hold due to the record $R(a, null)$. Clearly, $\mathcal{D}$ satisfies all possible TC statements under any semantics, since all records from the ideal database are also in the available database. Consider the query $Q(y) :- R(x, y)$ that projects R on the second argument. Then $Q^b(D^i) = \{ b \}$, while $Q^b(D^a) = \{ b, null \}$, which implies that Q is neither bag nor set complete over $\mathcal{D}$.

Theorem 2 shows that without unique dominance the unexpected situation can arise that all tables of a database are complete, according to the TC statements, yet some query is bag and set incomplete. Intuitively, a reason for this is that TC statements assert that a query result over the ideal database is *included* in a projection over the available database, while bag and set completeness require *equalities* to hold. Such equalities, however, may fail to hold because records in the available database may contain nulls where there are constants in the corresponding records in the ideal database.

The interplay of bag semantics and unique dominance can prevent this. Several copies of the same record in the result of a conjunctive query can be obtained from several combinations of records in the database. If the canonical TC statements hold under bag semantics, then for each such combination of records in the ideal database, there must be a corresponding combination in the available database. Morover, unique dominance ensures that two different combinations in the available database correspond to different combinations in the ideal database. This can be seen as the intuition behind Theorem 1.2.

The next theorem shows that the situation is different for TC statements under set semantics and set completeness of a query.

Theorem 3. *There exist a conjunctive query Q and an incomplete database $\mathcal{D}$ with unique dominance such that*

1. *$\mathcal{D}$ satisfies C^s_Q, the canonical TC statements for Q under set semantics;*
2. *$\mathcal{D}$ does* not *satisfy $\mathit{Compl}^s(Q)$, that is, set completeness of Q.*

Proof. Consider the query $Q(y) :- R(x, y)$, which projects R onto the second position. As Q has only the atom $A = R(x, y)$, there is a single canonical TC statement for Q, namely, $C_A = Compl(R(x, y); \{2\}; true)$.

Next, consider the partial database $\mathcal{D}$ consisting of $D^i = \{R(1, a), R(2, a)\}$ and $D^a = \{R(1, a), R(2, null)\}$. Clearly, $\mathcal{D}$ satisfies the principle of unique dominance. We easily check that $\mathcal{D}$ satisfies C_A, as $\pi^s_{\{2\}}(Q_{C_A}(D^i)) = \{a\} \subseteq \{a, null\} = \pi^s_{\{2\}}(R(D^a))$. However, $Q^s(D^i) = \{a\} \neq \{a, null\} = Q^s(D^a)$. Hence, Q is not set complete over $\mathcal{D}$.

5 Reasoning Problems and Preliminary Results

In this section we present four reasoning problems involving query completeness (QC) and table completeness (TC) and some preliminary results on them.

Problem 1: QC Characterization. Given a conjunctive query Q and a set of TC statements C, is C characterizing $Compl^\star(Q)$, that is, do we have $\mathcal{D} \models C$ if and only if $\mathcal{D} \models Compl^\star(Q)$ for all incomplete databases $\mathcal{D}$?

Preliminary Results. For bag completeness, the canonical TC statements under bag semantics are characterizing according to Theorem 1.2. An arbitrary set C is therefore characterizing for $Compl^b$(Q) if it is equivalent to C^b_Q. We have shown that in general for set completeness and set completeness modulo redundancy, query completeness cannot be characterized by a set of TC statements. An intuition is that for a tuple in the result of a such query there can be several derivations and for the two set semantics, just one of the many possible derivations is needed, which cannot be expressed by our TC statements, since a special kind of existential quantification would be needed.

The next problem is to find whether some canonical completeness statements can ensure query completeness.

Problem 2: TC-QC Entailment. Given a query Q, when and under which semantics do the canonical TC statements imply query completeness?

Preliminary Results. From Theorem 1.2 we know that the canonical statements under bag semantics entail bag completeness if we allow only incomplete databases satisfying unique dominance. It can be shown that under set semantics, they do not. Since query completeness under bag semantics entails query completeness under set semantics, the canonical statements under bag semantics entail QC under the two set semantics, provided we have unique dominance. According to Theorem 1.1, in the general case, canonical statements under set semantics entail set completeness modulo redundancy but, according to Theorem 3, may not entail set completeness proper. What holds for other combinations is an open question

If there are some TC statements that entail completeness of Q, a follow-up question is whether there exists a most general set of TC statements that entail completeness of Q, meaning a set that requires as little database completeness as possible.

Problem 3: Weakest Preconditions for TC-QC Entailment. Given a query Q, does there exist a set of TC statements C_0 such that $C_0 \models \mathit{Compl}^\star(Q)$, $\star \in \{ s, b, \mathit{sred} \}$, such that $C \models C_0$ for any other set C with this property?

Preliminary Results. For queries under bag semantics, it follows again from Theorem 1.2 that the canonical statements under bag semantics fulfil this requirement in the presence of unique dominance. For the two set semantics the problem is still open.

Finally, the most important problem is to decide whether a query can be answered completely, given knowledge about the completeness of parts of an incomplete database.

Problem 4: Deciding TC-QC Entailment. Given a query Q and a set of TC statements C, how can once check that whenever an incomplete database satisfies C it also satisfies $\mathit{Compl}^\star(Q)$, $\star \in \{ s, b, \mathit{sred} \}$.

Preliminary Results. For queries under bag semantics, TC-QC can be reduced to TC-TC entailement by Corollary 2. However, we do not know yet how to decide this. There are indications that it can be reduced to query containment under a combination of bag and set semantics over databases with null values.

For queries under set semantics modulo redundancy, the entailment $C \models C_Q^s$ is a sufficient condition. This problem, again, can be mapped to a problem of query containment under set semantics over databases with null values as a sufficient condition. It is open whether these sufficient conditions are also necessary.

6 Related Work

Motro [8] investigated query completeness as an aspect of query integrity. He introduced the notion of partially incomplete and incorrect databases as databases that can both miss facts that hold in the real world and contain facts that do not hold there. He described partial completeness in terms of *query completeness* (QC) statements under set semantics. To infer completeness of a given query from a set of queries known to be complete, he would search for a conjunctive rewriting of the given query in terms of the complete queries. This solution is correct, but not complete, as later results on query determinacy show [11].

Halevy [7] suggested *local completeness* statements, which we call table completeness (TC) statements, as an alternate formalism for asserting partial completeness of an incomplete database. The main problem he addressed was how to derive query completeness from table completeness (TC-QC reasoning). However, his approach led only to a decision procedure applicable to trivial cases.

Fan and Geerts [3] discussed the problem of query completeness in the presence of master data. Their work is not directly comparable to the one presented here because in addition to the different setting it always considers a database instance. In follow-up work, they considered incomplete data also in the form of missing but constrained attribute values [4], which they represented by *c-tables* [6].

Recently, Razniewski and Nutt picked up Levy's problem of TC-QC entailment over databases that can miss records [9,10]. They showed that TC-QC entailment is decidable for all languages of positive conjunctive queries used for formulating TC and

QC statements and analysed the complexity of the problem in detail, finding combined complexities ranging from PTIME to Π_2^P.

7 Conclusion

We have introduced the concept of incomplete databases with missing tuples and missing values, represented by SQL-style *null*s and identified as an important special case the one where unique record identifiers are always known, which leads to a property called *unique dominance.*

We introduced three different ways to define query completeness (QC) over an incomplete database, which are based on bag and set semantics of queries and take into account partiality of information in records with *null*s.

We generalised Levy's approach to describing complete parts of tables by table completeness (TC) statements to TC statements that describe completeness of projections of parts of tables. Depending on whether projections are performed under set or bag semantics, we defined two diffent semantics for these generalized TC statements.

We also generalized the canonical TC statements for queries from our previous work in such a way that they capture those projections of tables that are needed to answer a query. First results show how such generalized canonical TC statements can be used infer query completeness from other TC statements.

Finally, we have defined and discussed four reasoning problems: (1) finding a set of TC statements that characterize query completeness, (2) checking whether canonical TC statements under some semantic entail query completeness, (3) finding TC statements that are weakest preconditions query completeness, and (4) checking TC-QC entailment. For some of the problems we presented results while for others we sketched possible approaches.

In future work we aim to answer the open questions. Furthermore, we want to investigate the impact of schema constraints (keys, foreign keys, finite domains) on completeness reasoning.

References

1. S. Abiteboul, R. Hull, V. Vianu. Foundations of databases. *Addison-Wesley*, 1995.
2. E. F. Codd. Understing relations (installment #7). *FDT – Bulletin of ACM SIGMOD*, 7(3):23–28, 1975.
3. W. Fan, F. Geerts. Relative information completeness. *PODS*, 97–106, 2009.
4. W. Fan, F. Geerts. Capturing missing tuples and missing values. *PODS*, 169–178, 2010.
5. C. Farré, W. Nutt, E. Teniente, T. Urpí. Containment of conjunctive queries over databases with null values. *ICDT*, 389–403, 2007.
6. T. Imieliński, W. Lipski, Jr. Complete information in relational databases. *J. ACM*, 31:761–791, 1984.
7. A. Levy. Obtaining complete answers from incomplete databases. *VLDB*, 402–412, 1996.
8. A. Motro. tegrity = Validity + Completeness. *ACM TODS*, 14(4):480–502, 1989.
9. S. Razniewski, W. Nutt. Checking query completeness over incomplete data. *LID*, 2011.
10. S. Razniewski, W. Nutt. Completeness of queries over incomplete databases. *VLDB*, 2011.
11. L. Segoufin, V. Vianu. Views and queries: Determinacy and rewriting. *PODS*, 49–60, 2005.