# Completeness of Queries over SQL Databases

Werner Nutt
Free University of Bozen-Bolzano
Dominikanerplatz 3
39100 Bozen, Italy
nutt@inf.unibz.it

Simon Razniewski
Free University of Bozen-Bolzano
Dominikanerplatz 3
39100 Bozen, Italy
razniewski@inf.unibz.it

## ABSTRACT

Data completeness is an important aspect of data quality. We consider a setting, where databases can be incomplete in two ways: records may be missing and records may contain null values. We (i) formalize when the answer set of a query is complete in spite of such incompleteness, and (ii) we introduce table completeness statements, by which one can express that certain parts of a database are complete. We then study how to deduce from a set of table-completeness statements that a query can be answered completely.

Null values as used in SQL are ambiguous. They can indicate either that no attribute value exists or that a value exists, but is unknown. We study completeness reasoning for the different interpretations. We show that in the combined case it is necessary to syntactically distinguish between different kinds of null values and present an encoding for doing that in standard SQL databases. With this technique, any SQL DBMS evaluates complete queries correctly with respect to the different meanings that nulls can carry. We study the complexity of completeness reasoning and provide algorithms that in most cases agree with the worst-case lower bounds.

## Categories and Subject Descriptors

H.2.7 [**Database Management**]: Database Administration

## Keywords

Data Quality, Data Completeness, Metadata Management

## 1. INTRODUCTION

Decisions in business, politics and administration are taken on the basis of an ever increasing supply of information. To make the right decision, it is crucial to know how reliable the underlying data is. Often, data from diverse backgrounds are combined, which originate independently and according to different policies. As a consequence, the quality of the data may largely vary.

Aspects of data quality concern accuracy, currency, correctness, and similar issues [3]. In settings such as manual data insertion or data integration, completeness of data plays a key role [15]. Many approaches aim to improve data quality by inspecting and transforming concrete data. For instance, there is a wealth of techniques to detect and eliminate duplicate records. To deal with data completeness, such approaches have limited applicability: by inspecting data it is hard to detect whether or not something is missing.

Consider as a driving example the management of school data in a school district, which motivated the technical work reported here. The schools in the district are largely autonomous in the way they run their business: although the district provides a central database for administering data about students, teachers and the like, the schools can choose to what extent to use this system. As a consequence, on many topics data is incomplete, which becomes a problem when statistics about the schools are needed. While in principle, those statistics could be generated by queries over the database, the administration does not know whether or not it can trust the answers. In particular, if an item does not show up in a query result, or has the value null in an attribute, it is unclear whether that item does not have the property queried for in reality, or relevant data were just not submitted. The administration has some knowledge as to what data individual schools put into the central database. The question is how such metadata can be utilized to judge whether it is complete and thus can be trusted.

The first researcher to address this question was Motro who formalized completeness of databases and queries [14]. Halevy introduced statements, by which one can express that certain parts of a database are complete and raised the question how to use such statements to infer query completeness [12]. Recently, Razniewski and Nutt provided a general solution to this problem, including a technique to design algorithms and a comprehensive study of the complexity [16]. All this work considered only incompleteness in the form of missing records. In practice however, incompleteness in the form of *null* values is at least the same important. To take into account nulls we extend in this work previous formalisms by refining the granularity of completeness descriptions.

A problem with *null*s as used in standard SQL databases is their ambiguity, as those *null*s may mean both that an attribute value exists but is unknown, or that no value applies to that attribute. The established models of null values, such as Codd, v-, and c-tables [13], avoid this ambiguity by concentrating on the aspect of unknown values. In this work, we consider the ambiguous standard SQL *null* values [5], because those are the ones used in practice.

In this paper, we define a formal framework to study reasoning about the completeness of query answers over databases with *null* values and missing records. In Section 2, we introduce a school database as running example. Sections 3 formalizes databases with null values, incomplete databases and completeness statements. Section 4 presents the reasoning for simple, uniform meanings of null values, while in Section 5 we point out the limitations when combining the different meanings. Section 6 shows how the different meanings of nulls can be made explicit in standard SQL databases, while Sections 7 shows that reasoning is possible in that case. Section 8 discusses the reasoning for queries under bag semantics, Section 9 the complexity of reasoning, and Section 10 related work.

## 2. EXAMPLE: SCHOOL DATABASE

Our running example is that of a school database, which contains, inter alia, the following two tables:

- `student` (`name`, `classCode`, `homeTown`)
- `class` (`code`, `formTeacher`, `profile`)

The `student` table stores for each student the name, the class and the home town. Because we assume that the school is situated in a remote area, some students do not belong to any class but are taught at home. This would be indicated by a *null* value for the `classCode` attribute. It may also happen that there is no entry for the `homeTown` attribute, because the student did not provide this information or the secretary did not enter the data yet.

The `class` table stores the classes of the school. For each class, it stores its code, its form teacher and its profile, such as arts, science or similar. The `formTeacher` attribute may be *null*, because the decision of forming a class may have been taken before assigning the form teachers, or because the secretary may have forgotten to insert the form teacher. The profile attribute can be *null*, because some classes do not have a special profile.

## 3. FORMALISATION

In the following, we introduce standard notations about databases with null values and query evaluation, and introduce the notions of partial databases, query completeness and table completeness that we use in reasoning about completeness.

### 3.1 Relational Databases

A database *schema* is a set of relation symbols $\Sigma$ each with an arity. In the following, we assume the schema to be fixed. We assume an infinite set of constants *dom* including a special symbol $\perp$ for null values.

For a relation $R$ with arity $n$, an *atom* is an expression $R(t_1, \ldots, t_n)$, where $t_1 \ldots t_n$ are either elements of *dom* or variables. A database instance $D$ is a finite set of ground atoms. We sometimes refer to the atoms in an instance as facts. A *condition* is a set of atoms.

A *conjunctive query* is written as $Q(\bar{x}) \leftarrow L$, where $L$ is a condition and $\bar{x}$ is a tuple of variables, each occuring also in $L$. We call $L$ the *body* of $Q$, the variables in $\bar{x}$ the *distinguished variables* and the other variables in $L$ the *nondistinguished variables*. Given a conjunctive query $Q(\bar{x}) \leftarrow L$ and an instance $D$, an answer to $Q$ is a tuple $\alpha\bar{x}$, where $\alpha$ is an assignment of values to variables such that $\alpha L \subseteq D$. The set of all answers to $Q$ over $D$ is written as $Q^s(D)$. Similarly, we define $Q^b(D)$ as the bag of anwers that contains as many copies of a tuple as there are assignments returning it. We say that $Q$ is evaluated under set or bag semantics, respectively, if we refer to the set or bag of answers. We drop the superscripts $s$, $b$ if they are not important or clear from the context.

A conjunctive query is *linear*, if its body does not contain any relation symbol twice. It is *minimal*, if no atom can be removed from its body without changing the semantics of the query (cf. [4]).

### 3.2 Null Values

Nulls are a common form in which incompleteness is manifested in real-world databases. Null values mainly have two meanings:

- an attribute value exists, but is *unknown*;
- an attribute value does not exist, the attribute is thus *not applicable*.

In database theory, unknown values are represented by so-called *Codd nulls,* which are essentially existentially quantified first-order variables. A relation instance with Codd nulls, called a *Codd table*, represents the set of all regular instances that can be obtained by instantiating those variables with non-null values. [1]. For a conjunctive query $Q$ over an instance with Codd nulls, say $D_{\text{Codd}}$, one usually considers *certain answer* semantics [1]: the result set $Q_{\text{cert}}(D_{\text{Codd}})$ consists of those tuples that are in $Q(D')$ for every instantiation $D'$ of $D_{\text{Codd}}$. The set $Q_{\text{cert}}(D_{\text{Codd}})$ can be computed by evaluating $Q$ over $D_{\text{Codd}}$ while treating each occurrence of a null like a different constant and then dropping tuples with nulls from the result. Formally, using the notation

$$Q(D)^{\downarrow} := \{\, \bar{t} \in Q(D) \mid \bar{t} \text{ does not contain nulls} \,\}, \qquad (1)$$

this means $Q_{\text{cert}}(D_{\text{Codd}}) = Q(D_{\text{Codd}})^{\downarrow}$.

The null values supported by SQL ("SQL nulls" in short) have a different semantics than Codd nulls. Evaluation of first order queries follows a three-valued semantics with the additional truth value *unknown*. For a conjunctive query $Q$, we say that $y$ is a *join variable* if $y$ occurs at least twice in the body of $Q$ and a *singleton variable* otherwise. If $D_{\text{SQL}}$ contains facts with null values, then under SQL's semantics the result of evaluating $Q(\bar{x})$ over $D_{\text{SQL}}$ is

$$Q_{\text{SQL}}(D_{\text{SQL}}) = \{\, \alpha\bar{x} \mid \alpha \text{ maps no join variable to nulls} \,\}. \qquad (2)$$

To see this, note that a twofold occurrence of a variable $y$ is expressed in an SQL query by an equality between two attributes, which evaluates to *unknown* if a null is involved.

Recently, Franconi and Tessaris [11] have shown that the SQL way to evaluate queries over instances with nulls captures exactly the semantics of attributes that are not applicable. To make this more precise, suppose that $R$ is an $n$-ary relation with attribute set $X := \{A_1, \ldots, A_n\}$. If each attribute in an $R$-tuple can be null, then $R$ can be seen as representing for each $Y \subseteq X$ a relation $R_Y$ with attribute set $Y$. In this perspective, an instance of $R$ with tuples containing nulls represents a collection of $2^n$ instances of the relations $R_Y$, where a tuple $t$ belongs to the instance $R_Y$ iff the entries in $t$ for the attributes in $Y$ are not null. In other words, null values are padding the positions that do not correspond to attributes of $R_Y$.

*Example 1.* Consider the query $Q$ that asks for all classes whose form teacher is also form teacher of a class with arts as profile, which we write as $Q(c_1) \leftarrow \texttt{class}(c_1, t, p), \texttt{class}(c_2, t, \text{'arts'})$, and the instance $D = \{\, \texttt{class}(1a, \perp, \text{'arts'}) \,\}$. If we interpret $\perp$ as Codd-null, then $(1a) \in Q_{\text{Codd}}(D)$. If we evaluate $Q$ under the standard SQL semantics, we have that $(1a) \notin Q_{\text{SQL}}(D)$.

Suppose we know that class 1a has a form teacher. Then whoever the teacher of that class really is, the class has a teacher who teaches a class with arts as profile and the the first interpretation is correct. If no teacher exists, the second interpretation is correct.

Note that certain answer semantics and SQL semantics are not comparable in that the former admits more joins, while the latter allows for nulls in the query result. Later on we will show how for complete queries we can compute certain answers from SQL answers by simply dropping tuples with nulls.

We will say that a tuple with nulls representing an unknown but existing value is an *incomplete tuple*, since this nulls indicate the

| $D^i$ | |
|---|---|
| class | student |
| (1a, Smith, arts) | (John, 1a, Chester) |
| (2b, Rossi, $\bot$) | (Mary, $\bot$, Hampton) |
| | *(Paul, 2b, Westfield)* |

| $D^a$ | |
|---|---|
| class | student |
| (1a, Smith, arts) | (John, 1a, Chester) |
| (2b, Rossi, $\bot$) | (Mary, $\bot$, Hampton) |

**Table 1: Partial database with restricted facts**

| $D^i$ | |
|---|---|
| class | student |
| (1a, Smith, arts) | (John, 1a, Chester) |
| (2b, Rossi, science) | (Mary, 2b, Hampton) |
| | *(Paul, 2b, Westfield)* |

| $D^a$ | |
|---|---|
| class | student |
| (1a, Smith, arts) | (John, 1a, $\bot$) |
| (2b, Rossi, $\bot$) | (Mary, 2b, Hampton) |

**Table 2: Partial database with incomplete facts**

absence of existing values. We say that a tuple where nulls represent that no value exists is a *restricted tuple*, because only the not-null values in the tuple are related to each other. When modeling databases with null values, we will initially not syntactically distinguish between different kinds of null values and assume that some atoms in an instance contain the symbol $\bot$.

## 3.3 Partial Databases

When stating that data is incomplete, one must have a conceptual complete reference. We model partial databases in the style of Levy [12] as pairs $\mathcal{D} = (D^i, D^a)$ of database instances: $D^i$, the *ideal* state, with complete information, and $D^a$, the *available* state, which contains possibly less information. In general, both $D^i$ and $D^a$ may contain facts with nulls.

In an application, the state stored in a DBMS is the available state, which represents only a part of the facts that hold in reality. The facts holding in reality constitute the ideal state, which however is unknown. (Later on we will assume that we have some meta-information about the extent to which the available state captures the ideal state.)

We formalize that the available database contains less information than the ideal one using the concept of dominance: Let $R(\bar{s})$ and $R(\bar{t})$ be atoms that possibly contain nulls. Then $R(\bar{s})$ is *dominated* by $R(\bar{t})$, written $R(\bar{s}) \leq R(\bar{t})$, if $R(\bar{s})$ is the same as $R(\bar{t})$, except that $R(\bar{s})$ may have nulls where $R(\bar{t})$ does not. An instance $D$ is *dominated* by an instance $D'$, written $D \leq D'$, if each fact in $D$ is dominated by some fact in $D'$.

**Proposition 1 (Monotonicity)** *Let $Q$ be a conjunctive query and $D$, $D'$ be database instances with nulls. Suppose that $D$ is dominated by $D'$. Then $Q_{\text{cert}}(D) \subseteq Q_{\text{cert}}(D')$ and $Q_{\text{SQL}}(D) \leq Q_{\text{SQL}}(D')$.*

A *partial database* (or PDB for short) is a pair of database instances $(D^i, D^a)$ such that $D^a$ is dominated by $D^i$. We say that $\mathcal{D} = (D^i, D^a)$ is a *simple* partial database if $D^a \subseteq D^i$ and neither $D^i$ nor $D^a$ contains a null value. Completeness reasoning for simple partial databases has been studied in [16]. We say that $\mathcal{D}$ is a partial database *with restricted facts* if $D^a \subseteq D^i$. Note that in this case the ideal state may contain nulls and that every fact in the available state must appear in the same form in the ideal state. Thus, a null in the position of an attribute means that the attribute is not applicable and nulls are interpreted the way SQL does. The pair $(D^i, D^a)$ is a partial database *with incomplete facts* if $D^i$ does not contain any nulls and $D^a$ is dominated by $D^i$. In this case, there are no nulls in the ideal state, which means that all attributes are applicable, while the nulls in the available state indicate that attribute values are unknown. Therefore, those nulls have the same semantics as Codd nulls.

*Example 2.* Recall the school database from our running example, defined in Section 2. In Table 1 we see a partial database with restricted facts for this scenario. All null values appearing in the available database mean that no value exists for the corresponding attributes. The `class` table shows that no profile hass been assigned to class 2b and that Mary is an external student not belonging to any class.

In contrast, Table 2 shows a partial database with incomplete facts. Here, null values in the available database mean that a value exists but is unknown. So, class 1a has form teacher, but we do not know who. Class 2b has a profile, but we do not now which. John has a hometown, but we do not no from where he comes.

Observe that in both kinds of partial databases, some facts, such as the one about Paul being a student, can be missing completely.

In practice, null values of both meanings will occur at the same time, which may lead to difficulties if they cannot be distinguished.

## 3.4 Query Completeness

Data are accessed by posing queries. The core question we pursue in our work is whether a database has sufficient information to answer a query, that is, whether a query is *complete.* Intuitively, if we can infer from some meta-information that a query is complete, we know that the answer we receive over the available database at hand is the same as the one we would get when querying the ideal database. In other words, the available database contains all the data that is relevant for calculating the query answer, so one can trust the result that one gets.

We write $Compl^s(Q)$ and $Compl^b(Q)$, respectively, to indicate that $Q$ is complete under set or bag semantics. We now define when such a *query completeness* (QC) statement is satisfied by a partial database.

If $\mathcal{D} = (D^i, D^a)$ is a simple partial database (i.e., without nulls), then $\mathcal{D} \models Compl^s(Q)$ if and only if $Q^s(D^i) = Q^s(D^a)$, and $\mathcal{D} \models Compl^b(Q)$ if and only if $Q^b(D^i) = Q^b(D^a)$ (cf. [14]).

If $D$ contains nulls, then $Q^s(D)$ and $Q^b(D)$ depend on whether we interpret those as Codd or as SQL nulls.

Suppose $\mathcal{D}$ is a PDB with incomplete facts. Then nulls are interpreted as Codd nulls and queries are evaluated under certain answer semantics. While $D^a$ may contain nulls, $D^i$ does not and $Q_{\text{cert}}(D^i) = Q(D^i)$. We therefore define

$$\mathcal{D} \models_{\text{inc}} Compl^*(Q) \quad \text{iff} \quad Q^*(D^i) = Q^*_{\text{cert}}(D^a) \tag{3}$$

where $* \in \{ s, b \}$. That is, the tuples returned by $Q$ over $D^i$ are also returned over $D^a$ if nulls are treated according to certain answer semantics. Conversely, that every null-free tuple returned over $D^a$ is also returned over $D^i$ follows by mononoticity from the fact that $D^a \leq D^i$ (Proposition 1).

Suppose that $\mathcal{D}$ is a PDB with restricted facts. Then nulls are interpreted as SQL nulls and queries are evaluated under SQL semantics. For $* \in \{ s, b \}$ we define

$$\mathcal{D} \models_{\text{res}} Compl^*(Q) \quad \text{iff} \quad Q^*_{\text{SQL}}(D^i) = Q^*_{\text{SQL}}(D^a). \tag{4}$$

Again, the crucial part is that tuples returned by $Q$ over $D^i$ are also returned over $D^a$ if nulls are treated according to SQL semantics, while the converse inclusion holds due to monotonicity.

*Example 3.* The query $Q_{\text{art\_students}}(n) \leftarrow \texttt{student}(n, c, h), \texttt{class}(c, t, \text{'arts'})$ asks for the names of students in classes with arts as profile. Over $D^i$ in Table 2 it returns the singleton set $\{(\text{John})\}$ and over $D^a$ as well. Therefore, $Q_{\text{art\_students}}$ is complete over that partial database.

In contrast, $Q_{\text{homes}}(h) \leftarrow \texttt{student}(n, c, h)$ is not complete over this database, because it returns $\{(\text{Chester}), (\text{Hampton}), (\text{Westfield})\}$ over the ideal database but only $\{(\text{Hampton}), (\perp)\}$ over the available one.

## 3.5 Table Completeness

We use *table completeness* (TC) statements to specify that parts of a table are complete. A table completeness statement, written $Compl(R(\bar{s}); P; G)$, has three components: (i) a relational atom $R(\bar{s})$, (ii) a set of numbers $P \subseteq \{1, \ldots, arity(R)\}$, and (iii) a condition $G$. The numbers in $P$ are interpreted as attribute positions of $R$. For instance, if $R$ is the relation `student`, then $\{1, 3\}$ refers to the attributes `name` and `hometown`.

Let $C = Compl(R(\bar{s}); P; G)$ be a TC statement and $\mathcal{D} = (D^i, D^a)$ an incomplete database. An atom $R(\bar{u}) \in D^i$ is *constrained by* $C$ if there is an assignment $\alpha$ such that $\bar{u} = \alpha\bar{s}$, $R(\alpha\bar{s}) \in D^i$, and $\alpha G \subseteq D^i$. An atom $R(\bar{u}') \in D^a$ is an *indicator for* $R(\bar{u})$ *wrt* $C$ if $\bar{u}[P] = \bar{u}'[P]$, where $\bar{u}[P]$ is the projection of $\bar{u}$ onto the positions in $P$. We say that $C$ *is satisfied by* $\mathcal{D}$ if for every atom $R(\bar{u}) \in D^i$ that is constrained by $C$ there is an indicator $R(\bar{u}') \in D^a$.

Note that this type of TC statements extends the ones in [12] and [16] in that it allows one to state the completeness of projections of tables.

*Example 4.* In our school scenario, the TC statement

$$Compl(\texttt{student}(n, c, h); \{1, 2\}; \texttt{class}(c, t, \text{'arts'})) \tag{5}$$

states, intuitively, that the available database contains for all students of classes with arts as profile the name and the class. However, the student's hometown need not be present. Over the ideal database in Example 2, the fact $\texttt{student}(\text{John}, 1\text{a}, \text{Chester})$ is constrained by the statement (5). Any fact $\texttt{student}(\text{John}, 1\text{a}, \perp)$, $\texttt{student}(\text{John}, 1\text{a}, \text{Chester})$ or $\texttt{student}(\text{John}, 1\text{a}, \text{Clayton})$ in $D^a$ would be an indicator. In the database in Table 2 the first fact is present, and therefore Statement (5) is satisfied over it.

The semantics of TC statements can also be expressed using rule notation like the one that is used for instance for tuple-generating dependencies (TGDs) (see [7]). As a preparation, we introduce two copies of our signature $\Sigma$, which we denote as $\Sigma^i$ and $\Sigma^a$. The first contains a relation symbol $R^i$ for every $R \in \Sigma$ and the second contains a symbol $R^a$. Now, every incomplete database $(D^i, D^a)$ can naturally be seen as a $\Sigma^i \cup \Sigma^a$-instance. We extend this notation also to conditions $G$. By replacing every occurrence of a symbol $R$ by $R^i$ (resp. $R^a$), we obtain $G^i$ (resp. $G^a$) from $G$. Similarly, we define $Q^i$ and $Q^a$ for a query $Q$. With this notation, $(D^i, D^a) \models Compl(Q)$ iff $Q^i(D^i) = Q^a(D^a)$.

We associate to each TC statement $C = Compl(R(\bar{s}); P; G)$ a rule $\rho_C$. For instance, to Statement (5) we associate the rule

$$\texttt{class}^i(c, t, \text{'arts'}), \texttt{student}^i(n, c, h) \rightarrow \exists h'. \texttt{student}^a(n, c, h').$$

To simplify our notation, we assume that the projection positions $P$ are the first $k$ positions of $R$ and that $\bar{s}$ has the form $(\bar{s}', \bar{s}'')$, where $\bar{s}'$ has length $k$ and $\bar{s}''$ has length $arity(R) - k$. Then $\rho_C$ is

$$G^i, R^i(\bar{s}', \bar{s}'') \rightarrow \exists \bar{z}. R^a(\bar{s}', \bar{z}),$$

where $\bar{z}$ is a tuple of distinct fresh variables that has the same lenght as $\bar{s}''$. Clearly, for every TC statement $C$, an incomplete database satisfies $C$ in the sense defined above if and only if it satisfies the rule $\rho_C$ in the classical sense of rule satisfaction.

Note, that our definition of when a TC statement is satisfied takes into account null values. Regarding nulls in $D^a$, we treat nulls like non-null values and consider their presence sufficient to satisfy an existential quantification in the head of a TC rule.

Nulls in $D^i$, however, have to be taken into account when evaluating the body of a rule. Since nulls in the ideal database always represent the absence of a value, we always interpret the rules that we associated with TC statements under SQL semantics.

## 3.6 Reasoning

As usual, a set $C$ of TC statements *entails* completeness of a query $Q$ (we write $C \models Compl(Q)$) if every partial database that satisfies all statements in $C$ also satisfies $Compl(Q)$.

While TC statements are a natural way to describe completeness of available data ("These parts of the data are complete"), query completeness captures requirements for data quality ("For these queries we need complete answers"). Thus, in the following we investigate how to decide whether a set of TC statements entails query completeness (TC-QC entailment).

## 4. REASONING FOR SPECIFIC NULLS

In this section we discuss reasoning for databases where the meaning of nulls is unambiguous. In 4.1, we assume that nulls always mean that a value is missing but exists, while in 4.2, we assume that nulls mean that a value is inapplicable. For both cases we give decidable characterizations of TC-QC entailment. Moreover, we show that evaluation under certain answer and under SQL semantics lead to the same results for minimal complete queries.

## 4.1 Incomplete Facts

We suppose we are given a set of TC statements $C$ and a conjunctive query $Q$, which is to be evaluated under set semantics. We say that $C$ entails $Compl^s(Q)$ over PDBs with *incomplete facts*, written

$$C \models_{\text{inc}} Compl^s(Q), \tag{6}$$

iff for every such PDB $\mathcal{D}$ we have that

$$\mathcal{D} \models C \quad \text{implies} \quad \mathcal{D} \models_{\text{inc}} Compl^s(Q).$$

To decide property (6), we will derive a characterization that can be effectively checked.

To this end we introduce for every set $C$ of TC statements a transformation $T_C$ that, intuitively, maps an instance $D$ to the least informative instance $T_C(D)$ such that $(D, T_C(D)) \models C$. Let $C = Compl(\, R(\bar{s}', \bar{s}''); P; G\,)$ be a TC statement, where wlog $\bar{s}'$ consists of the terms in the positions $P$. We define the query $Q_C$ by the rule

$$Q_C(\bar{s}', \bar{\perp}) \leftarrow R(\bar{s}', \bar{s}''), G. \tag{7}$$

This means, given an instance $D$, the query $Q_C$ returns for every $\alpha$ satisfying the condition $R(\bar{s}', \bar{s}''), G$, a tuple $(\alpha\bar{s}', \bar{\perp})$ that consists of the projected part $\alpha\bar{s}'$ and is padded with $\perp$s for the positions projected out. We then define

$$T_C(D) := \{ R(\bar{t}) \mid \bar{t} \in Q_C(D) \} \tag{8}$$

and $T_{\mathcal{C}}(D) := \bigcup_{C \in \mathcal{C}} T_C(D)$.

Intuitively, for a database instance $D^i$ and a TC statement $C$, the function $T_C$ calculates the minimal information that any available database $D^a$ must contain in order that $(D^i, D^a)$ together satisfy $C$. Observe that every atom in $T_C(D)$ is an indicator for some $R(\bar{u})$ in $D$ wrt $C$. This is the case because every fact in $T_C(D)$ is created as an indicator for some fact in $D$ constrainted by $C$. Observe also that in general, $T_{\mathcal{C}}(D)$ may contain more facts than $D$, because several TC statments may constrain the same atom and therefore several indicators are produced.

*Example 5.* Consider the TC statement $C$ defined in Example 4 as $Compl(\texttt{student}(n, c, h);\ \{1, 2\};\ \texttt{class}(c, t, \text{'arts'}))$. The corresponding query is $Q_C(n, c, \bot) \leftarrow \texttt{student}(n, c, h), \texttt{class}(c, t, \text{'arts'})$. For the partial database in Table 2, $Q_C(D^i)$ is $\{(John, 1a, \bot)\}$ and hence $T_C(D^i) = \{\texttt{student}(John, 1a, \bot)\}$, which is the minimal information that any available database must contain to satisfy together with $D^i$ the TC statement $C$.

The following proposition formalizes the intuition about $T_{\mathcal{C}}$.

**Proposition 2** *Let $D$ be a database instance without nulls and let $\mathcal{D}_0$ be the partial database $(D, T_{\mathcal{C}}(D))$. Then*

1. *$T_{\mathcal{C}}(D)$ is dominated by $D$,*
2. *$\mathcal{D}_0$ is a PDB with incomplete facts, and*
3. *$\mathcal{D}_0 \models \mathcal{C}$.*

*Moreover, if $D'$ is another instance such that $(D, D')$ is a PDB with incomplete facts that satisfies $\mathcal{C}$, then $D'$ dominates $T_{\mathcal{C}}(D)$.*

Similar to a database instance, the body of a conjunctive query is a set of atoms, to which we can apply the transformation $T_{\mathcal{C}}$ if we view the variables as constants. The following characterization of TC-QC-entailment over PDBs with incomplete facts says that completeness of $Q$ wrt. $\mathcal{C}$ can be checked by evaluating $Q$ over $T_{\mathcal{C}}(L)$.

**Theorem 3** *Let $Q(\bar{x}) \leftarrow L$ be a conjunctive query and $\mathcal{C}$ be a set of table completeness statements. Then*

$$\mathcal{C} \models_{\text{inc}} Compl^s(Q) \quad \textit{iff} \quad \bar{x} \in Q_{\text{cert}}(T_{\mathcal{C}}(L)).$$

Proof. "⇒" By Proposition 2, $(L, T_{\mathcal{C}}(L))$ is a PDB with incomplete facts that satisfies $\mathcal{C}$. Thus, by assumption, $(L, T_{\mathcal{C}}(L)) \models_{\text{inc}} Compl^s(Q)$, which implies $Q^s(L) = Q^s_{\text{cert}}(T_{\mathcal{C}}(L))$. The identity from $L$ to $L$ is a satisfying assignment for $Q$ over $L$, from which it follows that $\bar{x} \in Q(L)$, and hence $\bar{x} \in Q_{\text{cert}}(T_{\mathcal{C}}(L))$.

"⇐" Suppose that $\bar{x} \in Q_{\text{cert}}(T_{\mathcal{C}}(L))$. We show that $\mathcal{C} \models_{\text{inc}} Compl^s(Q)$. Let $\mathcal{D} = (D^i, D^a)$ be a PDB with incomplete facts that satisfies $\mathcal{C}$ and suppose that $\bar{d} \in Q^s(D^i)$. We show that $\bar{d} \in Q^s_{\text{cert}}(D^a)$ (the converse holds because of the dominance of $D^a$ by $D^i$).

Given that there is an assignment $\delta$ such that $\delta L \subseteq D^i$ and $\delta\bar{x} = \bar{d}$ we construct an assignment $\delta'$ such that $\delta' L \subseteq D^a$ and $\delta'\bar{x} = \bar{d}$. To define $\delta'$, we specify how it maps atoms of $L$ to $D^a$.

Let $A$ be an atom in $L$. Since $\bar{x} \in Q_{\text{cert}}(T_{\mathcal{C}}(L))$, there is a homorphism $\theta$ from $L$ to $T_{\mathcal{C}}(L)$ such that $\theta\bar{x} = \bar{x}$. Let $B' = \theta A \in T_{\mathcal{C}}(L)$. By construction of $T_{\mathcal{C}}(L)$, there is a TC-statement $C = Compl(B;\ P;\ G)$ such that $(B, G) \subseteq L$ and $B'$ has been constructed as indicator for $B$ wrt. $C$. Since $\delta L \subseteq D^i$, we have $(\delta B, \delta G) \subseteq D^i$. Clearly, $\delta B$ is constrained by $C$ over $\mathcal{D}$. Since $\mathcal{D} \models \mathcal{C}$, there is an indicator atom $\tilde{B}$ for $\delta B$ in $D^a$. Defining that $\delta' A := \tilde{B}$, the mapping $\delta'$ proves that $\bar{d}$ is also in $Q(D^a)$ and hence that the query is complete. □

The idea of the theorem is the following: To check whether completeness of a query $Q$ is entailed by a set of TC statements $\mathcal{C}$, we perform a test over a prototypical database: Considering the body of the query as an ideal database, we test whether the satisfaction of the TC statements $\mathcal{C}$ implies that there is also enough information in any available database to return the tuple of the distinguished variables $\bar{x}$. If that is the case, then also for any other tuple found over an ideal database, there is enough information in the available database to compute that tuple again.

*Example 6.* Consider again the query from Example 3, which is $Q_{\text{art\_students}}(n) \leftarrow \text{student}(n, c, h), \text{class}(c, t, \text{'arts'})$. Suppose we are given TC statements $C_1 = Compl(\texttt{class}(c, t, p);\ \{1, 2, 3\};\ true)$ and $C_2 = Compl(\texttt{student}(n, c, h);\ \{1, 2\};\ \texttt{class}(c, t, p))$, which state that complete facts about all classes are in our database, and that for all students from art classes the name and the class attribute are in the database. When we want to find out whether $C_1$ and $C_2$ imply that query $Q_{\text{art\_students}}$ returns a complete answer, we proceed according to Theorem 3 as follows:

1. We take the body of the query $Q_{\text{art\_students}}$ as a prototypical test database: $L = \{\texttt{student}(n, c, h), \texttt{class}(c, t, p)\}$.
2. We apply the functions $T_{C_1}$ and $T_{C_2}$ to $L$ to generate the minimum information that can be found in any available database if the TC statements are satisfied: $T_{C_1}(L) = \{\texttt{class}(c, t, p)\}$ and $T_{C_2}(L) = \{\texttt{student}(n, c, \bot)\}$.
3. We evaluate $Q_{\text{art\_students}}$ over $T_{C_1}(L) \cup T_{C_2}(L)$. The result is $\{(n)\}$.

The tuple $(n)$ is exactly the distinguished variable of $Q_{\text{art\_students}}$. Therefore, we conclude that $C_1$ and $C_2$ entail query completeness under certain answer semantics.

We will discuss the complexity of reasoning in Section 9. At this point we only remark that the reasoning is in NP for conjunctive queries, since all that needs to be done is query evaluation, first of the TC rules in order to calculate $T_{\mathcal{C}}(L)$, second of $Q$, in order to check whether $\bar{x} \in Q(T_{\mathcal{C}}(L))$. Also, for conjunctive queries without self-joins the reasoning can be done in polynomial time.

So far we have assumed that nulls in the available database are treated as Codd nulls and that queries are evaluated under certain answer semantics. Existing DBMSs, however, implement the SQL semantics of nulls, which is more restrictive, as it does not allow for joins involving nulls, and thus leads to fewer answers. In the following we will show that SQL semantics gives us the same results as certain answer semantics for a query $Q$, if $Q$ is complete and minimal.

In analogy to "$\models_{\text{inc}}$", we define for a PDB with incomplete facts $\mathcal{D} = (D^i, D^a)$ that $\mathcal{D} \models_{\text{inc,SQL}} Compl^s(Q)$ if and only if $Q^s(D^i) = Q^s_{\text{SQL}}(D^a)^{\downarrow}$. Moreover, we write $\mathcal{C} \models_{\text{inc,SQL}} Compl^s(Q)$ if and only if $\mathcal{D} \models_{\text{inc,SQL}} Compl^s(Q)$ for all PDBs where $\mathcal{D} \models \mathcal{C}$. Intuitively, "$\models_{\text{inc,SQL}}$" is similar to "$\models_{\text{inc}}$", with the difference that queries over $D^a$ are evaluated as by an SQL database system.

We show that query completeness for this new semantics can be checked in a manner analogous to the one for certain answer semantics in Theorem 3. The proof is largely similar.

**Lemma 4** *Let $Q(\bar{x}) \leftarrow L$ be a conjunctive query and $\mathcal{C}$ be a set of table completeness statements. Then*

$$\mathcal{C} \models_{\text{inc,SQL}} Compl^s(Q) \quad \textit{iff} \quad \bar{x} \in Q_{\text{SQL}}(T_{\mathcal{C}}(L)).$$

Now, suppose that the conjunctive query $Q$ is minimal (cf. [4]). Then $Q$ returns a result over $T_{\mathcal{C}}(L)$ only if each atom from $L$ has an indicator in $T_C(L)$. The next lemma shows that it does not matter whether the nulls in $T_C(L)$ are interpreted as SQL or as Codd nulls.

**Lemma 5** *Let $C$ be a set of TC statements and $Q(\bar{x}) \leftarrow L$ be a minimal conjunctive query. Then*

$$\bar{x} \in Q_{\mathrm{SQL}}(T_C(L)) \quad \textit{iff} \quad \bar{x} \in Q_{\mathrm{cert}}(T_C(L)).$$

Combining Theorem 3 and Lemmas 4 and 5 we conclude that for minimal conjunctive queries that are known to be complete, it does not matter whether one evaluates them under certain answer or under SQL semantics.

**Theorem 6** *Let $\mathcal{D} = (D^i, D^a)$ be an incomplete database with incomplete facts, let $C$ be a set of TC statements, and let $Q(\bar{x}) \leftarrow L$ be a minimal conjunctive query. If $C \models_{\mathrm{inc}} Compl^s(Q)$ and if $\mathcal{D} \models C$ then $Q^s_{\mathrm{cert}}(D^a) = Q^s_{\mathrm{SQL}}(D^a)^{\downarrow}$.*

It follows that for complete queries we also get a complete query result when evaluating them over standard SQL databases.

*Example 7.* Consider again the query $Q_{\mathrm{art\_students}}$ from Example 3, where $Q_{\mathrm{art\_students}}(n) \leftarrow \mathtt{student}(n, c, h), \mathtt{class}(c, t, \text{'arts'})$, and the TC statements $C_1$ and $C_2$ from Example 6 that entailed query completeness over PDBs with incomplete facts. Since $Q_{\mathrm{art\_students}}$ has no self-joins it is clearly minimal, and hence over the available database of any PDB that satisfies $C_1$ and $C_2$ we can evaluate it under set semantics and will get a complete query result.

## 4.2 Restricted Facts

We now move to PDBs with restricted facts. Recall that in this case a null in a fact indicates that an attribute is not applicable. Accordingly, a PDB with restricted facts is a pair $(D^i, D^a)$ where both the ideal and the available database may contain nulls, and where the available is a subset of the ideal database ($D^a \subseteq D^i$).

Again, we suppose that we are given a set of TC statements $C$ and a conjunctive query $Q(\bar{x}) \leftarrow L$, which is to be evaluated under set semantics. Similar to the case of incomplete facts, we say that *$C$ entails $Compl^s(Q)$ over PDBs with restricted facts*, written

$$C \models_{\mathrm{res}} Compl^s(Q), \tag{9}$$

iff for every such PDB $\mathcal{D}$ we have that

$$\mathcal{D} \models C \quad \text{implies} \quad \mathcal{D} \models_{\mathrm{res}} Compl^s(Q).$$

We will derive a characterization of (9) that can be effectively checked. We reuse the function $T_C$ defined in Equation (8).

**Proposition 7** *Let $D$ be an instance that may contain nulls and let $\mathcal{D}_1 = (D \cup T_C(D), T_C(D))$. Then*

1. *$\mathcal{D}_1$ is a PDB with restricted facts;*
2. *$\mathcal{D}_1 \models C$.*

*Moreover, if $D'$ is another instance such that $(D \cup D', D')$ is a PDB with restricted facts that satisfies $C$, then $D'$ dominates $T_C(D)$.*

In contrast to databases with incomplete facts, nulls can now appear in the output of queries over the ideal database, and therefore must not be ignored in query answers over the available database. Recent results in [11] imply that for queries over databases with restricted facts, evaluation according to SQL's semantics of nulls returns correct results.

The characterization of completeness entailment is different now because $Q$'s body $L$ is no more a prototypical instance for $Q$ to retrieve an answer $\bar{x}$. Since the ideal database may now contain nulls, we must consider the case that variables in $L$ are mapped to $\perp$ when $Q$ is evaluated over $D^i$.

We first present a result for *boolean* queries, that is, for queries where the tuple of distinguished variables $\bar{x}$ is empty, and for *linear* (or self-join free) queries, that is, queries where no relation symbol occurs more than once.

A variable $y$ in a query $Q(\bar{x}) \leftarrow L$ is a *singleton* variable, if it appears only once in $L$. Recall that only singleton variables can be mapped to $\perp$ when evaluating $Q$ under SQL semantics. Let $L^{\perp}$ and $\bar{x}^{\perp}$ be obtained from $L$ and $\bar{x}$, respectively, by replacing all singleton variables with $\perp$.

**Theorem 8** *Let $Q(\bar{x}) \leftarrow L$ be a boolean or linear conjunctive query and $C$ be a set of table completeness statements. Then*

$$C \models_{\mathrm{res}} Compl^s(Q) \quad \textit{iff} \quad \bar{x}^{\perp} \in Q_{\mathrm{SQL}}(T_C(L^{\perp})).$$

The theorem reduces completeness reasoning in the cases above to conjunctive query evaluation. We conclude that deciding TC-QC entailment wrt databases with restricted facts is in PTIME for linear and NP-complete for arbitrary boolean conjunctive queries.

For general conjunctive queries, which may have distinguished variables, evaluating $Q$ over a single test database obtained from $L$ is not enough. We can show, however, that it is sufficient to consider all cases where singleton variables in $L$ are either null or not. A *null version* of $L$ is a condition obtained from $L$ by replacing some singleton variables with $\perp$. We represent null versions of $L$ as instantiations $\gamma L$, where $\gamma$ is a substitution that replaces some singleton variables of $L$ with $\perp$ and is the identity otherwise.

**Theorem 9** *Let $Q(\bar{x}) \leftarrow L$ be a conjunctive query. Then the following are equivalent:*

- *$C \models_{\mathrm{res}} Compl^s(Q)$;*
- *$\gamma\bar{x} \in Q_{\mathrm{SQL}}(T_C(\gamma L))$, for every null version $\gamma L$ of $L$.*

The theorem says that instead of just one prototypical case, we have to consider several now, because query evaluation for databases with nulls is more complicated: while the introduction of nulls makes the satisfaction of TC statements and the query evaluation more difficult, it also creates more possibilities to retrieve null as a result (see [10]).

The above characterisation can be checked by a $\Pi^P_2$ algorithm: in order to verify that containment does not hold, it suffices to guess one null version $\gamma L$ and then show that $\gamma\bar{x}$ is not in $Q(T_C(\gamma L))$, which is an NP task.

## 5. AMBIGUOUS NULLS

So far we have assumed that nulls have one of two possible meanings, standing for unknown or for non-existing values. In this section we discuss completeness reasoning in the presence of one syntactic null value, which can have three possible meanings, the previous two plus indeterminacy as to which of those two applies. This is the typical usage of nulls in SQL.

We model PDBs for this case as pairs $\mathcal{D} = (D^i, D^a)$, where both instances, $D^i, D^a$, may contain $\perp$ and each tuple in $D^a$ is dominated by a tuple in $D^i$. We assume that queries are evaluated as in SQL, since we cannot tell which nulls are Codd-nulls and which not. For a query $Q$ and $* \in \{ s, b \}$ we define

$$\mathcal{D} \models_{\mathrm{ambg}} Compl^*(Q) \quad \text{iff} \quad Q^*_{\mathrm{SQL}}(D^i) = Q^*_{\mathrm{SQL}}(D^a). \tag{10}$$

Different from the case where nulls stand for unknown values, we may not drop nulls in the query result over the available database, because they might carry information (absence of a value).

We observe that without further restrictions on the PDBs, for many queries there is no way to conclude query completeness from table completeness.

**Proposition 10** *There exists a PDB $\mathcal{D}$ with ambiguous nulls and a query $Q$, such that $\mathcal{D}$ satisfies any set of TC statements but $\mathcal{D}$ does not satisfy $Compl^s(Q)$.*

Proof. Let $\mathcal{D}$ be with $D^i = \{ \texttt{student}(\text{Mary}, \text{2a}, \text{Chester}) \}$ and $D^a = \{ \texttt{student}(\text{Mary}, \text{2a}, \text{Chester}), \texttt{student}(\text{Mary}, \bot, \text{Chester}) \}$. Clearly, $\mathcal{D}$ satisfies all possible TC statements, because $D^a \subseteq D^i$. But query $Q_{\text{classes}}(c) \leftarrow \texttt{student}(n, c, h)$ is not complete over $\mathcal{D}$, because $Q^s(D^i) = \{ (2a) \}$ while $Q^s(D^a) = \{ (2a), (\bot) \}$. □

Inspecting $\mathcal{D}$ in the proof above more closely, we observe that the two facts in $D^a$ are dominated by the same fact in the ideal database. Knowing that, we can consider the second fact in $D^a$ as redundant: it does not add new information about Mary. This duplicate information leads to the odd behaviour of $\mathcal{D}$ wrt completeness: while all information from the ideal database is also in the available database, $Q(D^a)$ contains an additional fact with a null.

Sometimes, such duplicates occur naturally, e.g., when data from different sources is integrated. In other scenarios, however, redundancies are unlikely because objects are identified by keys, and only non-key attributes may be unknown or non-applicable.

In a school database, it can happen that address or birth place of a student are unknown. In contrast, it is hard to imagine that one may want to store a fact $\texttt{student}(\bot, \bot, \text{Chester})$, saying that there is a student with unknown name and class living in Chester.

Keys alone, however, are still not sufficient:

*Example 8.* Suppose we are given a partial database with $D^i = \{\texttt{student}(\text{Mary}, \text{2a}, \text{Chester}), \texttt{student}(\text{Paul}, \text{2a}, \text{Hampton})\}$ and $D^a = \{ \texttt{student}(\text{Mary}, \text{2a}, \text{Chester}), \texttt{student}(\textit{Paul}, \bot, \text{Hampton}) \}$. Observe that there are no redundant tuples in $D^a$. The TC statement $Compl(\texttt{student}(n, c, h); \{ 2 \}; \textit{true})$, which says that all classes from $D^i$ are also in $D^a$, is satisfied over this PDB. One might believe that over a PDB satisfying this statement the query $Q_{\text{classes}}$, defined above, is complete, as it is the case for PDBs with incomplete facts or with restricted facts. However, query evaluation returns that $Q^s_{\text{classes}}(D^i) = \{ (2a) \}$ while $Q^s_{\text{classes}}(D^a) = \{ (2a), (\bot) \}$.

The problem with ambiguous nulls is that while all information needed for computing a query result may be present in the available database, it is not clear how to treat a null in the query answer. If it represents an unknown value, we can discard it because the value will still be there explicitly. But if it represents that no value exists, it should also show up in the query result.

Therefore, we conclude that one should disambiguate the meaning of null values. In the next section we propose how to do this in an SQL database.

## 6. MAKING NULL SEMANTICS EXPLICIT

Nulls in an available database can express three different statements about a value: absence, presence with the concrete value being unknown, and indeterminacy which of the two applies. As seen in Section 5, this ambiguity makes reasoning impossible. To explicitly distinguish between the three meanings of nulls in an SQL database, we present an approach that adds an auxiliary boolean attribute to each attribute that possibly has nulls as values.

| `student` | | |
|---|---|---|
| `name` | `...` | `code` |
| Sara | | 2a |
| John | | $\bot_{\text{uk}}$ |
| Mary | | $\bot_{\text{n/a}}$ |
| Paul | | $\bot_{\bot}$ |

| `student` | | | |
|---|---|---|---|
| `name` | `...` | `hasCode` | `code` |
| Sara | | *true* | 2a |
| John | | *true* | $\bot$ |
| Mary | | *false* | $\bot$ |
| Paul | | $\bot$ | $\bot$ |

**Table 3: Making the semantics of nulls explicit**

*Example 9.* Consider relation $\texttt{student}(\texttt{name}, \texttt{code}, \texttt{hometown})$. Imagine a student John for whom the attribute `code` is null because John attends a class, but the information was not entered into the database yet. Imagine another student Mary for whom `code` is null because Mary is an external student and does not attend any class. Imagine a third student Paul for whom `code` is null because it is unknown whether or not he attends a class. We mark the different meanings of nulls by symbols $\bot_{\text{uk}}$ (unknown but existing value), $\bot_{\text{n/a}}$ (not applicable value) and $\bot_{\bot}$ (indeterminacy), but remark that in practice, in an SQL database, all three cases would be expressed using syntactically identical null values.

We can distinguish them, however, if we add a boolean attribute `hasCode`. For John, the value of `hasCode` would be *true*, expressing that the tuple for John has a code value, which happens to be unknown, indicated by the $\bot$ for `code`. For Mary, `code` would have the value *false*, expressing that the attribute `code` is not applicable. For Paul, the `hasCode` attribute itself would be $\bot$, expressing that nothing is known about the actual value. Table 3 shows a `student` instance with explicit types of null, on the left using three nulls, on the right with a single null and the auxiliary attribute.

In general, for an attribute `attr` where we want to disambiguate null values, we introduce a boolean attribute `hasAttr`. We refer to `hasAttr` as the *sign* of `attr`, because it signals whether a value exists for the attribute, no value exists, or whether this is unknown.

Note that if `hasAttr` is *false* or $\bot$, then `attr` must be $\bot$. This can be enforced by an SQL check constraint.

As seen earlier, in general SQL semantics does not fully capture the semantics of unknown nulls as it may miss some certain answers. We will show in Theorem 12, that our encoding can be exploited to compute answer sets for complete queries by joining attributes with nulls according to SQL semantics and then using the signs to drop tuples with unknown and indeterminate nulls.

## 7. REASONING FOR DIFFERENT NULLS

In the previous section we showed how to implement a syntactic distinction of three different meanings of null values in SQL databases. In this section we discuss how to reason with these three different nulls.

An instance $D$ with the three different kinds of nulls represents an infinite set of instances $D'$ that can be obtained from $D$ by (i) replacing all occurrences of $\bot_{\text{uk}}$ with concrete values and (ii) replacing all occurrences of $\bot_{\bot}$ with concrete values or with $\bot_{\text{n/a}}$.

As usual, the set of *certain answers* of a query $Q$ over $D$ consists of the tuples that are returned by $Q$ over all such $D'$ and is denoted as $Q_{\text{cert}}(D)$.

It is easy to see that a tuple $\bar{d}$ is in $Q_{\text{cert}}(D)$ iff the only nulls in $\bar{d}$ are $\bot_{\text{n/a}}$ and there exists an assignment $\alpha$ such that (i) $\bar{d} = \alpha\bar{x}$, (ii) $\alpha L \subseteq D$, (iii) $\alpha$ does not map join variables to $\bot_{\text{n/a}}$ or $\bot_{\bot}$, and (iv) no two occurrences of a join variable are mapped to different occurences of $\bot_{\text{uk}}$. Intuitively, this means that we have to treat $\bot_{\text{uk}}$ as Codd null and the other nulls as SQL nulls.

We say that a partial database $\mathcal{D} = (D^i, D^a)$ contains *partial facts* if (i) the facts in $D^i$ may contain the null $\bot_{\text{n/a}}$, (ii) the facts in $D^a$ may contain all three kinds of nulls, and (iii) each fact $R(\bar{t}) \in D^a$ is *dominated* by a fact $R(\bar{t}')$ in $D^i$ in the sense that for any position $p$

- if $\bar{t}[p] = \bot_{\text{n/a}}$, then also $\bar{t}'[p] = \bot_{\text{n/a}}$,
- if $\bar{t}[p] = \bot_{\text{uk}}$, then $\bar{t}'[p]$ is a value from the domain *dom*,
- if $\bar{t}[p] = \bot_{\bot}$, then $\bar{t}'[p]$ is $\bot_{\text{n/a}}$ or in *dom*,
- if $\bar{t}[p] = d$ for a value $d \in dom$, then also $\bar{t}'[p] = d$.

We then say that a query is complete over a database $\mathcal{D} = (D^i, D^a)$ with partial facts, if $Q(D^i) = Q_{\text{cert}}(D^a)$, and write $\mathcal{D} \models_{3\bot} \mathit{Compl}(Q)$.

Satisfaction of TC-statements is not affected by these changes, as $D^i$ contains only nulls $\bot_{\text{n/a}}$, which indicate restricted facts that can be treated according to SQL semantics.

*Example 10.* Consider the available database $D^a$ that contains the three facts $\texttt{class}(1\text{a}, \bot_{\text{uk}}, \text{'arts'})$, $\texttt{class}(2\text{b}, \bot_{\text{n/a}}, \text{'arts'})$ and $\texttt{class}(3\text{c}, \bot_{\bot}, \text{'arts'})$. Also, consider the query from Example 1 that asks for all classes whose form teacher is also form teacher of an arts class, written as $Q(c_1) \leftarrow \texttt{class}(c_1, t, p_1), \texttt{class}(c_2, t, \text{'arts'})$.

Then similar to before, the only tuple in $Q_{\text{cert}}(D^a)$ is $(1a)$, because since the teacher of that class is unknown but existing, it holds in any complete database that the class 1a has a teacher that also teaches an arts class (1a again). The tuples 2b and 3c do not show up in the result, because the former has no form teacher at all ($\bot_{\text{n/a}}$), while the latter may or may not have a form teacher.

A first result is that TC-QC entailment over PDBs with partial facts is equivalent to entailment over PDBs with restricted facts:

**Theorem 11** *Let $Q$ be a conjunctive query and $C$ be a set of TC statements. Then*

$$C \models_{3\bot} \mathit{Compl}^s(Q) \quad \textit{iff} \quad C \models_{\text{res}} \mathit{Compl}^s(Q).$$

Proof. (Sketch) "$\Rightarrow$" Trivial, because PDBs with restricted facts are PDBs with partial facts that contain only the null value $\bot_{\text{n/a}}$.

"$\Leftarrow$" Assume, $C \not\models_{3\bot} \mathit{Compl}^s(Q)$. Then there is a PDB with partial facts $\mathcal{D}$ such that $\mathcal{D} \models C$, but $\mathcal{D} \not\models_{3\bot} \mathit{Compl}^s(Q)$. We construct a PDB $\mathcal{D}_0$ with restricted facts that also satisfies $C$, but does not satisfy $\mathit{Compl}^s(Q)$. Let $D_0^a = D^a[\bot_{\text{uk}}/\bot_{\text{n/a}}, \bot_{\bot}/\bot_{\text{n/a}}]$ be the variant of $D^a$ where $\bot_{\text{uk}}$ and $\bot_{\bot}$ are replaced by $\bot_{\text{n/a}}$, and let $D_0^i = D^i \cup D_0^a$.

The additional facts in $D_0^i$ do not lead to violations of TC statements, since they are dominated by facts in $D^i$, thus, $\mathcal{D}_0 \models C$. However, $Q(D_0^a) \subseteq Q(D^a)$, since changing nulls to $\bot_{\text{n/a}}$ makes query evaluation more restrictive, and $Q(D^i) \subseteq Q(D_0^i)$ due to monotonicity. Hence, $Q(D_0^a) \subsetneq Q(D_0^i)$, that is, $\mathcal{D}_0 \not\models_{\text{res}} \mathit{Compl}^s(Q)$. □

Also, we define the query evaluation $Q(D)^{\Downarrow}$ as $Q(D)$ without all tuples containing $\bot_{\text{uk}}$ or $\bot_{\bot}$.

Similar to a database with incomplete facts only, it holds that query answering for minimal queries that are complete does not need to take into account certain answer semantics but can safely evaluate the query using standard SQL semantics:

**Theorem 12** *Let $\mathcal{D} = (D^i, D^a)$ be an incomplete database with partial facts, $Q$ be a minimal conjunctive query and $C$ be a set of table completeness statements. If $C \models_{3\bot} \mathit{Compl}^s(Q)$ and $\mathcal{D} \models C$ then $Q^s_{\text{cert}}(D^a) = Q^s_{\text{SQL}}(D^a)^{\Downarrow}$.*

## 8. QUERIES UNDER BAG SEMANTICS

Bag semantics is the default semantics of SQL queries, while set semantics is enabled with the DISTINCT keyword. As the next example shows, for relations without keys reasoning about query completeness under bag semantics may not be meaningful.

*Example 11.* Consider the partial database with incomplete facts $\mathcal{D} = (D^i, D^a)$, where $D^i = \{\texttt{student}(\text{Mary}, 2\text{a}, \text{Chester})\}$ and $D^a = \{\texttt{student}(\text{Mary}, 2\text{a}, \text{Chester}), \texttt{student}(\text{Mary}, \bot, \text{Chester})\}$. Since it is a priori not possible to distinguish whether the fact containing $\bot$ is redundant, the boolean query $Q() \leftarrow \texttt{student}(n, c, h)$ that is just counting the number of students is not complete, because the redundant tuple in the available database leads to a miscount.

As tuples with nulls representing unknown values can introduce redundancies, we require that keys are declared for PDBs with incomplete facts, with one ambiguous null or with partial facts. Only for partial databases with restricted facts keys are not necessary, because there the available database is always a subset of the ideal one and hence no redundancies can appear.

Formally, for a relation $R$ with arity $n$, a *key* is a subset of the attribute positions $\{1, \ldots, n\}$. Wlog, we assume that the key attributes are the first $k(R)$ attributes, where $k$ is a function from relations to natural numbers. An instance $D$ *satisfies the key* of a relation $R$, if (i) no nulls appear in the key positions of facts and (ii) no two facts have the same key values, that is, if for all $R(\bar{t})$, $R(\bar{t}') \in D$ it holds that $\bar{t}[1..k(R)] = \bar{t}'[1..k(R)]$ implies $\bar{t} = \bar{t}'$, where $\bar{t}[1..k(R)]$ denotes the restriction of $\bar{t}$ to the positions $1..k(r)$.

Table completeness statements that do not talk about all key attributes of a key are not useful for deciding the entailment of query completeness under bag semantics, because, intuitively, they cannot assure that the right multiplicity of information is in the available database. We say that a TC statement $\mathit{Compl}(R(\bar{x}); P; G)$ is *key-preserving*, if $\{1..k(R)\} \subseteq P$. In the following, we only consider TC statements that are key-preserving.

We develop a characterization for TC-QC entailment that is similar to the one for set semantics. However now, we need to ensure that over a prototypical database not only query answers but assignments are preserved, because a query answer tuple can be produced by different assignments. So if an assignment is missing, the multiplicity of a tuple in the result is incorrect. As a consequence, a set of TC statements may entail completeness of a query $Q$ for set semantics, but not for bag semantics.

*Example 12.* The relation $\texttt{lnCourse(student,language)}$ stores the language courses that students take. Consider the query $Q_{\text{nr\_for\_french}}(n) \leftarrow \texttt{lnCourse}(n, l), \texttt{lnCourse}(n, \text{'french'})$, which counts for each student that attends French, how many language courses he attends. Under set semantics, $Q_{\text{nr\_for\_french}}$ is complete if $D^a$ contains all facts about French courses, which is expressed by $C_{\text{french}} = \mathit{Compl}(\texttt{lnCourse}(n, \text{'french'}); \{1, 2\}; \mathit{true})$. To test completeness for set semantics, we apply $T_C$ to the query body $L$, which results in $T_C(L) = \{\texttt{lnCourse}(n, \text{'french'})\}$, since the first body atom is not constrained by $C_{\text{french}}$. Evaluating $Q_{\text{nr\_for\_french}}$ over $T_C(L)$ returns $(n)$, which shows set completeness.

But this does not entail that $Q_{\text{nr\_for\_french}}$ is complete under bag semantics. The PDB $(L, T_C(L))$ is a counterexample: it satisfies $C$ and we can evaluate $Q_{nr\_for\_french}$ over $L$ two times, while over $T_C(L)$ just once. If Paul takes French and Spanish according to $D^i$, it is clearly not sufficient to only have the fact about French in $D^a$ when we want to count how many courses Paul takes!

We therefore modify the test criterion in Theorem 8 in two ways.

For a query $Q(\bar{x}) \leftarrow L$, the tuple $\bar{w}$ of *crucial variables* consists of the variables that are in $\bar{x}$ or occur in key positions in $L$. For any two assignments $\alpha$ and $\beta$ that satisfy $L$ over a database $D$, we have that $\alpha$ and $\beta$ are identical if they agree on $\bar{w}$. Thus, the crucial variables determine both, the answers of $Q$ and the multiplicities with which they occur. We associate to $Q$ the query $\bar{Q}(\bar{w}) \leftarrow L$ that has the same body as $Q$, but outputs all crucial variables. Consequently,

$Q$ is complete under set semantics if and only if $\bar{Q}$ is complete under set semantics. The first modification of the criterion will consist in testing $\bar{Q}$ instead of $Q$.

A direct implication of the first modification is that we need not consider several null versions $\gamma L$ of $L$ as in Theorem 9. The reason for doing so was that a null $\perp = \alpha x$ in the ouput of $Q$ over $\gamma L$ could have its origin in an atom $\gamma A$ in $\gamma L$ such that $x$ does not occur in $A$, but another variable, say $y$ is instantiated to $\perp$. Now, the query $\bar{Q}$ passes the test for set completeness only if an atom in $L$ is mapped to an atom with the same key values. Thus, a variable $x$ cannot be bound to a null $\perp = \gamma y$. Hence it suffices to consider just the one version $L^\perp$ where all singleton variables are mapped to null. By the same mapping, $\bar{w}^\perp$ is obtained from $\bar{w}$.

The second modification is due to the possibility that several TC statements constrain one fact in $D^i$ und thus $T_C$ generates several indicators. Since we assumed TC statements to be key-preserving, these indicators all agree on their key positions. However, in some non-key position one indicator may have a null while another one has a non-null value. So, $T_C(L^\perp)$ may not satisfy the keys. This can be repaired by "chasing" $T_C(L^\perp)$ (cf. [1]).

The function *chase* takes a database $D$ with nulls as input and merges any two $R$-facts $A'$, $A''$ that have the same key values into one $R$-fact $A$ as follows: the value of $A$ at position $p$, denoted $A[p]$ is $A'[p]$ if $A'[p] \neq \perp$ and is $A''[p]$ otherwise. Clearly, if $C$ is key-preserving and $D$ satisfies the keys, then $chase(T_C(D))$ also satisfies the keys. Intuitively, *chase* condenses information by applying the key constraints. Obviously, *chase* runs in polynomial time.

We now are ready for our characterization of completeness entailment under bag semantics, which is similar, but slightly more complicated than the one in Theorem 8.

**Theorem 13** *Let $Q(\bar{x}) \leftarrow L$ be a conjunctive query and $C$ be a set of key-preserving TC statements. Then*

$$C \models_{3\perp} Compl^b(Q) \quad \textit{iff} \quad \bar{w}^\perp \in \bar{Q}(chase(T_C(L^\perp))).$$

Since the criterion holds for incomplete databases with three different nulls, it holds also for the special cases where only one type of null values is present (restricted or incomplete facts).

Notably, it also holds for partial databases with one ambiguous null, because when keys are present and TC statements guarantee that all mappings are preserved, no additional nulls can show up in the query result.

## 9. COMPLEXITY OF REASONING

We now discuss the complexity of inferring query completeness from table completeness. We define TC-QC$_\star$ as the problem of deciding whether under $\star$-semantic for all partial databases $\mathcal{D}$ it holds that $\mathcal{D} \models C$ implies that $\mathcal{D} \models Compl(Q)$. We will find that for all cases considered in the paper, the complexity of reasoning is between NP and $\Pi_2^P$:

**Theorem 14 (Complexity Bounds)**

1. TC-QC$^s_{\text{inc}}$ *is NP-complete;*
2. TC-QC$^s_{\text{res}}$ *is NP-hard and in $\Pi_2^P$;*
3. TC-QC$^s_{3\perp}$ *is NP-hard and in $\Pi_2^P$;*
4. TC-QC$^b_{3\perp}$ *is NP-complete.*

Proof. NP-hardness in all four cases can be shown by a reduction of containment of Boolean conjunctive queries, which is known to be NP-complete [4]. We sketch the reduction for (1)–(3), the one for (4) being similar. Suppose we want to check whether $Q() \leftarrow L$ is contained in $Q'() \leftarrow L'$. Let $P$ be a new unary relation. Consider the query $Q_0() \leftarrow P(a), L$ and the TC statement $C_0 = Compl(P(a); \{1\}; L')$. Let $C$ consist of $C_0$ and the statement that $R$ is complete for every relation $R$ in $L$. Then it follows from Theorems 3, 8 and 11 that $C \models_* Compl^s(Q)$, where $* \in \{\text{inc}, \text{res}, 3\perp\}$, if and only if $P(a) \in T_C(L)$, $P(a) \in T_C(L^\perp)$, and $P(a) \in T_C(L^\perp)$, respectively. The latter three conditions hold iff $P(a) = T_{C_0}(P(a), L)$, which holds iff $Q$ is contained in $Q'$.

Problem 1 is in NP, because according to Theorem 3 to show that the entailment holds, it suffices to construct $T_C(L)$ by guessing assignments that satisfy sufficiently many TC statements in $C$ over $L$, and to guess an assignment that satisfies $Q$ over $T_C(L)$ such that the tuple $\bar{x}$ is returned.

Problem 2 is in $\Pi_2^P$, because according to the characterization in Theorem 9, to show that entailment does not hold, it suffices to guess one null version $\gamma L$ of the body of $Q$ and show that $\gamma\bar{x}$ is not in $Q(chase(T_C(\gamma L)))$, which is an NP task.

Problem 3 is in $\Pi_2^P$ for the same reason.

Problem 4 is in NP, because we do not consider different nullversions of $L$ but only one. The remaining argument is the same as for Problem 1, since one needs to show that $\bar{x}^\perp$ is in $T_C(L^\perp)$, which is an NP task. □

|  | Query semantics |  |
|---|---|---|
| Partial database class | set semantics | bag semantics & databases with keys |
| no nulls | NP-complete | NP-complete |
| incomplete facts | NP-complete | NP-complete |
| restricted facts | NP-hard, in $\Pi_2^P$ | NP-complete |
| partial facts | NP-hard, in $\Pi_2^P$ | NP-complete |

**Table 4: Complexity of TC-QC entailment**

Reasoning becomes easier for special cases of queries:

**Theorem 15 (Special Cases)** *Let $* \in \{\text{inc}, \text{res}, 3\perp\}$. Then*

1. TC-QC$^s_*$ *and* TC-QC$^b_*$ *are in PTIME for linear queries;*
2. TC-QC$^s_*$ *is NP-complete for boolean queries.*

Proof. Regarding Claim 1, the most critial case is $* = \text{res}$. For linear queries under bag semantics, observe that the criterion in Theorem 13 can be checked in polynomial time. First, there is only one choice to map an atom in a query $Q_C$ to an atom in $L^\perp$ (the one with the same relation). Second, $chase(T_C(L^\perp))$ can be computed in polynomial time. Lastly, the evaluation of $\bar{Q}$ over the chase result is in PTIME, because an atom in $\bar{Q}$ can be mapped in only one way.

Note that for linear queries under set semantics, we only need to consider one null version $L^\perp$ because a binding for an output term can only come from one position.

The lower bounds of Claim 2 follow from Theorem 14, the upper bounds from Theorem 14 for inc, and from Theorems 8 and 11 for res and $3\perp$, since evaluation of conjunctive queries is in NP. □

In Table 4 we summarize our complexity results for TC-QC entailment over databases with nulls. Notably, if we have keys then under bag semantics the complexity does not increase with respect to databases without null values. For queries under set semantics, it remains open whether the complexity of reasoning increases from NP to $\Pi_2^P$ for databases with restricted facts and with 3 null values.

## 10. RELATED WORK

Since the introduction of null values in relational databases [5], there has been a long debate about their semantics and the correct implementation. In particular, the implementation of nulls in

SQL has led to wide criticism and numerous proposals for improvement (for a survey, see [19]). Much work has been done on the querying of incomplete databases with missing but existing values [2, 17], while only recently, Franconi and Tessaris showed that SQL correctly implements null values that stand for inapplicable attributes [11]. It was observed early on that different syntactic null values in databases would allow to capture more information [6], but these ideas did not reach application.

Query completeness over incomplete databases was first studied by Motro [14]. He investigated query completeness as an aspect of query integrity, and 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, but do not contain null values. He described partial completeness in terms of *query completeness* (QC) statements under set semantics. To infer completeness of a 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 [18].

Halevy [12] suggested *local completeness* statements, which we call table completeness (TC) statements, as an alternate formalism for asserting partial completeness of an incomplete database without nulls. 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 [8] 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 existing values [9], which they represented by *c-tables* [13].

Recently, Razniewski and Nutt picked up Levy's problem of TC-QC entailment over databases that can miss records [16]. They showed that TC-QC entailment is decidable if conjunctive queries with comparisons are used for formulating TC and QC statements and analysed the complexity of the problem in detail for queries under bag semantics, set semantics and aggregate queries, finding complexities ranging from PTIME to $\Pi_2^P$.

## 11. CONCLUSION

Although null values are an important aspect of incompleteness in relational databases, they were not supported by formalisms for reasoning about data completeness. We generalized partially complete databases, table completeness and query completeness statements as in [12, 16] so that one can describe incompleteness in the form of both missing records and missing values.

Our approach addresses two challenges. First, if a table contains nulls, it may be that not entire rows, but only the columns of some rows are complete. To describe such a situation, we refined table completeness statements so that they can refer to projections of parts of tables. Second, null values as used in practice are ambiguous (unknown, not applicable, indeterminate). To resolve such ambiguities, we proposed an encoding for SQL databases that makes diffent kinds of nulls explicit. We then developed techniques to infer the completeness of conjunctive queries from such refined statements for different semantics of nulls and assessed the complexity of the reasoning task.

While SQL's query evaluation is generally not correct for nulls that represent missing values, we showed that for a minimal complete query correct query answers can be calculated from the SQL query result by dropping tuples with unknown and indeterminate nulls.

In this work, we focussed on conjunctive queries without built-in predicates like inequalities or disequalities. Treatment of built-in is orthogonal to the one of nulls, so that our results can be extended to them, using the techniques in [16]. Similarly, as aggregate queries are computed on top of non-aggregate queries, our results can also be extended in this direction.

## Acknowledgements

This work was partially supported by the ESF project 2-299-2010 "SIS - Wir verbinden Menschen", and the European commission FP7 project ICT-2009.4.2 "TERENCE".

## 12. REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] Serge Abiteboul, Paris C. Kanellakis, and Gösta Grahne. On the representation and querying of sets of possible worlds. *Theor. Comput. Sci.*, 78(1):158–187, 1991.

[3] Carlo Batini and Monica Scannapieco. *Data Quality: Concepts, Methodologies and Techniques*. Springer, 2006.

[4] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, pages 77–90, 1977.

[5] Edgar F. Codd. Understanding relations (installment #7). *FDT – Bulletin of ACM SIGMOD*, 7(3):23–28, 1975.

[6] Edgar F. Codd. Missing information (applicable and inapplicable) in relational databases. *SIGMOD Record*, 15(4):53–78, 1986.

[7] Ronald Fagin, Phokion Kolaitis, Renée Miller, and Lucian Popa. Data exchange: Semantics and query answering. In *Proc. ICDT*, pages 207–224, 2003.

[8] Wenfei Fan and Floris Geerts. Relative information completeness. In *PODS*, pages 97–106, 2009.

[9] Wenfei Fan and Floris Geerts. Capturing missing tuples and missing values. In *PODS*, pages 169–178, 2010.

[10] Carles Farré, Werner Nutt, Erneste Tenient, and Toni Urpí. Containment of conjunctive queries over databases with null values. In *ICDT*, pages 389–403, 2007.

[11] Enrico Franconi and Sergio Tessaris. On the logic of SQL nulls. In *AMW*, 2012.

[12] Alon Y. Halevy. Obtaining complete answers from incomplete databases. In *Proc. VLDB*, pages 402–412, 1996.

[13] Tomasz Imieliński and Witold Lipski, Jr. Incomplete information in relational databases. *J. ACM*, 31, 1984.

[14] Amihai Motro. Integrity = Validity + Completeness. *ACM TODS*, 14(4):480–502, 1989.

[15] Felix Naumann, Johann C. Freytag, and Ulf Leser. Completeness of integrated information sources. *Inf. Syst.*, 29, 2004.

[16] Simon Razniewski and Werner Nutt. Completeness of queries over incomplete databases. In *VLDB*, 2011.

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

[18] Luc Segoufin and Victor Vianu. Views and queries: Determinacy and rewriting. In *PODS*, pages 49–60, 2005.

[19] Ron van der Meyden. Logical approaches to incomplete information: A survey. In *Logics for Databases and Information Systems*, pages 307–356, 1998.