Tables with Nulls and Functional Dependencies: Explanation and Quality of Query Answers

Dominique Laurent, Nicolas Spyratos

Tables with Nulls and Functional Dependencies: Explanation and Quality of Query Answers*

Dominique Laurent[1][0002−7264−9576] and Nicolas Spyratos[2][0002−3432−8608]

[1] ETIS Lab.-ENSEA, CY Cergy Paris Univ., CNRS, F-95000 Cergy-Pontoise, France
dominique.laurent@u-cergy.fr
[2] LISN Lab.-University Paris-Saclay, CNRS, F-91405 Orsay, France
nicolas.spyratos@lri.fr

Abstract. Several applications today deal with tables that are the result of merging other tables coming from different sources (as when recording the results of collaborative work or when merging tables during data staging in data warehouses). Such tables usually have missing values (also called *nulls*) and/or contain data that do not respect given constraints(such as key constraints). In this paper we study the influence of nulls and/or inconsistent data on the answers to queries to such tables by (a) providing to the user explanations regarding the expected presence (or absence) of certain tuples in the answer and (b) by defining measures for assessing the quality of query answers.

Keywords: Inconsistent data · Query answering · Explanations · Data quality.

1 Introduction

In a traditional relational database, each table is assumed to be consistent before users can query it but, in several applications today, tables are the result of merging two or more other tables coming from different sources. Such tables usually have missing values (also called *nulls*) and/or contain data that do not respect given constraints - such as key constraints. This is especially true when a user collects data coming from web sources and merges them in a single table T. In such a setting, it is difficult if not impossible for the user to know the constraints imposed on data in each source. Given a set FD of constraints over T, extracting from T data that are consistent with FD is then achieved using some query tool. In this paper we propose such a tool.

Three main issues (among others) arising when querying such tables are how to extract consistent answers addressed to possibly inconsistent tables; how to help users explain the expected presence or absence of certain tuples in the consistent answer to a query; and how to give users a measure of the quality of a query answer. We describe below briefly these issues, placing them in the context of related work.

* Work conducted while the second author was visiting at FORTH Institute of Computer Science, Crete, Greece (https://www.ics.forth.gr/)

Consistent query answering. Consider the table $T = \{es,\ es'd,\ e's'\}$ over universe $U = \{Emp, Sal, Dept\}$, and the functional dependency $Emp \rightarrow Sal$ stating that an employee cannot have more than one salary. T is inconsistent as tuples 1 and 2 violate the dependency $Emp \rightarrow Sal$. However, if we ask the SQL query Q : select Emp, Sal from T, it makes sense to return the set $\{e's'\}$ as the answer. Indeed, there is no reason to reject this answer as it is a consistent answer because it satisfies the dependency $Emp \rightarrow Sal$. In other words, an inconsistent table may contain consistent parts (i.e. some useful information) which can be extracted through queries.

This kind of query answering, known as *consistent query answering*, has attracted a lot of attention since the 1990s and continues to be an important subject of research today ([2, 3, 14, 20]). It is thus not possible to review all related approaches here, and we refer to [5] for a detailed survey covering the topic. We emphasize that two main approaches to consistent query answering have emerged: the approach by 'table repairs' [2, 20], for tables with functional dependencies but without nulls, and more recently a more general approach without table repairs for tables with functional dependencies *and* nulls [12, 13].

Explanation of query answers. With the growing popularity of big data, many users with a variety of backgrounds seek to extract high level information from data-sets. A major challenge is then to develop tools to assist users in explaining observed query outputs.

In this paper we study explanations related to the presence of inconsistent tuples in the computation of a query answer. For instance, in our previous example of employees, salaries and departments, if we ask the SQL query Q : select Sal from T the answer will be empty. Given that the consistent query answer over employee and salary contains the tuple $e's'$, the user may wonder why the answer over salary is empty. The explanation is that, although $e's'$ is consistent (that is e' is assigned the only salary s'), s' is also involved in an inconsistent tuple (namely es' as e is assigned two salaries).

Related work in this domain was initiated by Halpern and Pearl in [9, 10], where basic motivation and formal definitions of causality are presented and discussed in detail. In particular, in [10], an explanation of φ is defined as *a minimal elaboration of events that suffice to cause φ even in the face of uncertainty about the actual situation.* This generic definition corresponds in our work to explaining why a tuple t is or not in the consistent answer to a given query Q, in the simple case where there is no uncertainty about the actual situation. Our way of explaining the answers to queries is also related to *data lineage* [15]. According to this work, 'explaining' query answers is achieved based on the database content and on the expression of the query. However, in this work the authors do not consider inconsistent databases as we do. Dealing with causality and explanation in the presence of inconsistent data has been presented in [6], relying on the notion of database repairs, which we do not use in our work.

We also note that another typical scenario is when a user integrates data sets, computes some statistics, and then seeks an explanation why certain values

are or are *not* outliers ([16, 17]). Such kind of explanations lies however out of the scope of the present paper.

Quality of query answers. The answer to a query addressed to a consistent table in a relational database, comes from tuples that are consistent and that are *assumed* to be true (a basic assumption in relational databases). In contrast, the answer to a query addressed to a possibly inconsistent table may be computed from inconsistent tuples. For instance, in our previous example of employees, salaries and departments, the answer comes from two inconsistent tuples (es and $es'd$) and one consistent tuple ($e's'$). In this paper, we introduce various quality measures such as percentages of consistent or inconsistent tuples in the table expressing the 'influence' of inconsistent data on the answers to queries.

Regarding related work, inconsistency measurement has been addressed in [8], based on three-valued propositional logic. In this setting, the authors propose 10 distinct possible measures, and discuss their relevance in the context of databases. Moreover, it is shown in [8] that, contrary to our approach, most quality measures lead to non tractable computations. In [11], the authors address the issue of 'approximate inference of functional dependency', and to do so they define quality measures for assessing the 'quality' of functional dependencies in a given table. Roughly, these measures are based on the number of *conflicting pairs* of tuples of the form (xy, xy'), for a given functional dependency $X \to Y$. Contrary to the more generic case of [8], the authors of [11] argue that computing their measures is in $\mathcal{O}(N.\log(N))$ where N is the number of tuples in T, thus resulting in a tractable complexity. To compare this work with our work, we notice that although the contexts are similar (namely a possibly inconsistent table with functional dependencies), the goals of the two approaches are different since ours considers the problem of consistent query answering and in [11], the authors address the issue of functional dependency inference.

As will be seen later in the paper, relating explanations and data quality in presence of inconsistent data is a relevant issue. This task, first identified in [4], remains to be further investigated.

Summarizing the above discussion, we address the issues of query answer explanation and quality measures for query answers in a table with nulls and functional dependencies. In order to study these issues, we need to know which tuples of the table are consistent or inconsistent, on the one hand, and which tuples are false or true, on the other hand.

The reason why such knowledge is necessary is twofold. First, we want to define the answer to a query to contain only tuples that are true and consistent (as in relational databases); in this way if a user seeks an explanation such as why certain tuples are (or are *not*) in an answer, we will be able to justify the presence or absence of the tuples in question by telling the user that the tuples are false or inconsistent. Second, we want to be able to provide measures for measuring the quality of data in the table and in the query answers. As we saw in the previous example of employees, salaries and departments, a consistent answer to a query may contain tuples computed from inconsistent tuples. Therefore we want to define measures of the 'influence' of such tuples on the data in the table and on

the data of a query answer. For example, the higher the ratio of consistent to true tuples in the table, the higher the quality of the data in the table; and similarly, the higher the percentage of tuples in the answer computed from consistent tuples in the table, the higher the 'confidence' in the query answer.

The remainder of this paper is organized as follows: in Sections 2 and 3, we recall from [13] our definitions of m-Chase and of consistent query answer, respectively; in Section 4 we present our approach to explanations of query answers; in Section 5 we present our approach to quality measures for tables with nulls and functional dependencies, as well as for consistent query answers. In Section 6 we offer some concluding remarks and perspectives of our work.

2 The m-Chase Algorithm

2.1 Notation and Basic Definitions

As in the relational model, we consider a universe $U = \{A_1, \ldots, A_n\}$ in which every attribute A_i is associated with a set of atomic values called the domain of A_i and denoted by $dom(A_i)$. We call *relation schema* (or simply *schema*) any nonempty sub-set of U and we denote it by the concatenation of its elements.

A *tuple* t over U is a partial function from U to $\bigcup_{A \in U} dom(A)$ such that, if t is defined over A then $t(A)$, also denoted $t.A$, belongs to $dom(A)$. The domain of definition of t is called the *schema* of t, denoted by $sch(t)$. Tuples in our approach satisfy the *First Normal Form* [19], in the sense that each tuple component is an atomic value. A tuple t is denoted by the concatenation of its values: $t = a_{i_1} \ldots a_{i_k}$ means that for every $j = 1, \ldots, k$, $t.A_{i_j} = a_{i_j}$, where a_{i_j} is in $dom(A_{i_j})$, and $sch(t) = A_{i_1} \ldots A_{i_k}$. We define a *table* over U to be a *finite* set of tuples over U (therefore duplicates are not allowed), and we note that as tuples are partial functions, tables may contain nulls.

Given a table T, we denote by $\mathcal{T}$ the set of all tuples built up from values in T. Queries are issued against T and consistent answers are obtained from $\mathcal{T}$. For every relation schema X, we denote by $\mathcal{T}(X)$ the set of all tuples in $\mathcal{T}$ with schema X: $\mathcal{T}(X) = \{t \in \mathcal{T} \mid sch(t) = X\}$. For every A in U, the set of all values from $dom(A)$ occurring in T is called the *active domain of* A, denoted by $adom(A)$, and we let $\mathcal{AD} = \bigcup_{A \in U} adom(A)$. In other words, $\mathcal{AD}$ is the set of all values appearing in T.

Given a tuple t, for every nonempty sub-set S of $sch(t)$ the restriction of t to S, is denoted by $t.S$. In this work, tables over universe U are associated with a fixed set of functional dependencies FD. A functional dependency is an expression of the form $X \to A$ where X is a schema and A an attribute not in X. A table T is said to satisfy $X \to A$ if for all t and t' in T such that $t.XA$ and $t'.XA$ contain no nulls, if $t.X = t'.X$ then $t.A = t'.A$.

2.2 The m-Chase Algorithm

In order to characterize a tuple of a given table T as consistent/inconsistent and as true/false, we define a modified version of the classical chase algorithm

[7, 19]. We recall that, given a table T with nulls and a set FD of functional dependencies, the chase algorithm constructs an 'extended' table denoted by $chase(T)$ as follows:

for all t and t' such that there exists $X \to A$ in FD and $t.X = t'X$
  if $t.A$ and $t'.A$ are *distinct* domain values, **then** *fail*
  else if $t.A = a$ and $t'.A$ is null **then** assign a to $t'.A$

If the chase algorithm succeeds then all tuples in the resulting table $chase(T)$, are consistent and are *assumed* to be true (a basic assumption in relational databases). However, if the chase algorithm fails then we do not know which tuples are consistent and which are inconsistent.

To cope with this problem, a modified version of the chase algorithm was introduced in [13], which allows to *always* know which tuples are consistent and which are inconsistent. This algorithm, called *m-Chase* works as follows: distinct values $t.A$ and $t'.A$ of the chase algorithm above do *not* provoke a failure; instead, such values are accumulated in a set thus creating m-tuples (i.e., tuples in which each component can be a set of values instead of a single value).

In our introductory example where $T = \{es, es'd, e's'\}$, running the usual chase algorithm with the functional dependency $Emp \to Sal$ would result in failure since the tuples $t = es$ and $t' = es'd$ violate the dependency $Emp \to Sal$. In contrast, our algorithm, called *m-Chase* will put the two values $t.Sal$ and $t'.Sal$ in a set to create what we call an *m-tuple* $(e)(ss')(d)$ (to be defined shortly), where concatenation of values between parentheses denotes a set. For example, (e) stands for $\{e\}$, (ss') stands for $\{s, s'\}$ and (d) stands for $\{d\}$. The idea is to accumulate in a set all values of an attribute violating a dependency. As we shall see shortly, based on the set of m-tuples returned by the m-Chase we can compute the sets of true/false and consistent/inconsistent tuples in polynomial time; and based on these sets we can give explanations of query answers and we can also define quality measures as explained earlier.

We now recall from [13] the basic formalism on which algorithm *m-Chase* relies. First the notion of *multi-valued tuples*, or *m-tuples*, extends that of tuples in the sense that an m-tuple associates every attribute A with a possibly empty *sub-set* of $adom(A)$, instead of a *single* value from $adom(A)$.

Definition 1. *A* multi-valued tuple σ *over universe* U*, or m-tuple, is a function from* U *to the cross product* $\mathsf{X}_{A \in U} \mathcal{P}(adom(A))$*, where* $\mathcal{P}(adom(A))$ *is the power set of* $adom(A)$*. The set of all attributes* A *such that* $\sigma(A) \neq \emptyset$*, is called the* schema of σ*, denoted by* $sch(\sigma)$*. Given* σ *and a sub-set* X *of* $sch(\sigma)$*, the restriction of* σ *to* X*, denoted* $\sigma(X)$*, is the m-tuple defined by* $(\sigma(X))(A) = \sigma(A)$ *for every* A *in* X *and* $(\sigma(X))(A) = \emptyset$ *for any* A *not in* X*.*

Given an m-tuple σ*, the set* $\mathsf{tuples}(\sigma)$ *denotes the set of all tuples* t *such that* $sch(t) = sch(\sigma)$ *and for every* A *in* $sch(t)$*,* $t.A$ *belongs to* $\sigma(A)$*.* □

Given an m-tuple σ, the set $\sigma(A)$ is denoted by the concatenation of its elements between parentheses, and σ is denoted by the concatenation of all $\sigma(A)$ such that $\sigma(A) \neq \emptyset$. Moreover, $\sigma \sqsubseteq \sigma'$ denotes the 'component-wise inclusion' of σ in σ', that is $\sigma \sqsubseteq \sigma'$ holds if for every $A \in sch(\sigma)$, $\sigma(A) \subseteq \sigma'(A)$.

Algorithm 1 The m-Chase Algorithm

Input: A table T over U and a set FD of functional dependencies over U.
Output: An m-table denoted by Σ^*.

```
1: Σ* := {σ_t | t ∈ T}// σ_t is the m-tuple such that σ_t(A) = {t.A} for A ∈ sch(t)
2: change := true
3: while change = true do
4:    change := false
5:    for all σ and σ' in Σ* do
6:       for all X → A in FD such that XA ⊆ sch(σ) and XA ⊆ sch(σ) do
7:          if tuples(σ(X)) ∩ tuples(σ'(X)) ≠ ∅ then
8:             apply the m-Chase rule to σ and σ'
9:             change := true
10: return Σ*
```

We call *m-table* over U any finite set of m-tuples over U. For all σ and σ' in an m-table Σ, and $X \to A$ such that $XA \subseteq sch(\sigma)$ and $XA \subseteq sch(\sigma')$, the following rule called m-Chase rule generalizes the chase rule.

- m-Chase rule: Let $\sigma_1 = \sigma \cup \sigma'(A)$ and $\sigma'_1 = \sigma' \cup \sigma(A)$
  Case of $\sigma_1 \sqsubseteq \sigma'_1$: replace σ with σ'_1, and remove σ_1
  Case of $\sigma'_1 \sqsubseteq \sigma_1$: replace σ' with σ_1, and remove σ'_1
  Otherwise: replace σ and σ' with σ_1 and σ'_1, respectively.

As shown in Algorithm 1 our algorithm consists in applying the m-Chase rule whenever $\mathsf{tuples}(\sigma(X)) \cap \mathsf{tuples}(\sigma'(X)) \neq \emptyset$ until no further transformation is possible. The output is an m-table Σ^*, and it has been shown in [13] that this algorithm always terminates and that the partition semantics of tuples in $\mathcal{T}$ (as introduced in [18] and extended in [12, 13]), is defined based on Σ^* as follows.

Definition 2. *For every t in $\mathcal{T}$:*

1. *t is* true *if there exists σ in Σ^* and q in $\mathsf{tuples}(\sigma)$ such that t is a sub-tuple of q. The set of all true tuples is denoted by $\mathsf{True}(\mathcal{T})$.*
2. *t is* false *if t is not true. The set of all false tuples is denoted by $\mathsf{False}(\mathcal{T})$.*
3. *t is* inconsistent *if there exists σ in Σ^* such that $\mathsf{tuples}(\sigma(sch(t))) = \{t\}$. The set of all inconsistent tuples is denoted by $\mathsf{Inc}(\mathcal{T})$.*
4. *t is* consistent *if t is true and not inconsistent. The set of all consistent tuples is denoted by $\mathsf{Cons}(\mathcal{T})$.* □

As shown in [13], the computation of Σ^* is in $\mathcal{O}(|\Sigma^*|^3.\delta^2)$, where δ is the maximal cardinality of the components of m-tuples in Σ^*, which is precisely the maximum number of A-values associated with X-values when $X \to A$ is a functional dependency in FD. As Algorithm 1 shows that $|\Sigma^*| \leq |T|$, we state that the computation of Σ^* is in $\mathcal{O}(|T|^3.\delta^2)$, i.e., *polynomial in the size of T*.

Example 1. In the context of our introductory example, where $T = \{es, es'd, e's'\}$ is a table defined over $U = \{Emp, Sal, Dept\}$ with the functional dependency $Emp \to Sal$, Σ^* is built up according to the following steps:

Step 1: Σ^* is first set to $\{(e)(s), (e)(s')(d), (e')(s')\}$.
Step 2: Considering $\sigma = (e)(s)$ and $\sigma' = (e)(s')(d)$, the m-tuples $\sigma_1 = (e)(ss')$ and $\sigma'_1 = (e)(ss')(d)$ are generated. Since $\sigma_1 \sqsubseteq \sigma'_1$, $\Sigma^* = \{(e)(ss')(d), (e')(s')\}$.
Step 3: As a new execution of the while-loop line 3 does not change Σ^*, the algorithm returns $\Sigma^* = \{(e)(ss')(d), (e')(s')\}$.

Therefore, $\mathsf{True}(\mathcal{T})$ contains esd, $es'd$, $e's'$ and all their sub-tuples. Hence, $\mathsf{False}(\mathcal{T}) = \{e'sd, e's, e'd\}$. Moreover, $\mathsf{Inc}(\mathcal{T}) = \{esd, es'd, es, es', sd, s'd, s, s'\}$, and thus $\mathsf{Cons}(\mathcal{T}) = \{ed, e's', e, e', d\}$. □

It turns out from Definition 2 that a tuple t can be either *true* or *false*, and that true tuples are either *consistent* or *inconsistent*. However, *false* tuples are neither consistent nor inconsistent. Notice that, since we only focus on true tuples, the consistency of false tuples is irrelevant in this work. Membership of t to $\mathsf{True}(\mathcal{T})$ or $\mathsf{False}(\mathcal{T})$ on the one hand, and to $\mathsf{Cons}(\mathcal{T})$ or $\mathsf{Inc}(\mathcal{T})$ on the other hand, is referred to as the *status* of t. Thus a tuple can have one of the following status: true and consistent, true and inconsistent or false. Referring to Example 1, esd is true and inconsistent, $s'd$ is true and inconsistent and $e's$ is false.

3 Consistent Query Answering

3.1 Query Syntax

The queries Q that we consider in this work have the form of a usual SQL query:

$$Q : \texttt{select}\ X\ \texttt{from}\ T\ [\texttt{where}\ \Gamma]$$

in which the `where` clause specifies an optional selection condition Γ. The set of all attributes occurring in Γ is called the *schema of* Γ, denoted by $sch(\Gamma)$; and the attribute set $X \cup sch(\Gamma)$ is called the *schema of* Q, denoted by $sch(Q)$.

A selection condition Γ is a well-formed formula built up using connectors $\neg$, $\vee$, $\wedge$ and atomic comparisons of the forms: $A\,\theta\,a$ or $A\,\theta\,A'$, where θ is a comparison predicate, A and A' are attributes, and a is in $dom(A)$. Given Γ, we denote by $Sat(\Gamma)$ the set of all tuples in $\mathcal{T}(sch(\Gamma))$ satisfying Γ, where the notion of satisfaction follows the usual rules in FO logics. For example, the tuple $t = abcc$ over scheme $ABCD$ is in $Sat(\Gamma)$ for $\Gamma = ((A = a') \vee (C = D)) \wedge (B = b)$.

We emphasize that, contrary to most existing approaches to consistent query answering [3, 14, 20], our approach is not restricted to deal with conjunctive queries, since disjunctive selection conditions are allowed. The *consistent answer* to a query Q in our approach relies on the notion of *consistency with respect to a selection condition* as defined below.

Definition 3. *Given a table T over U, and Γ a selection condition, a tuple t such that $sch(\Gamma) \subseteq sch(t)$ is said to be* consistent with respect to Γ *if there exists σ in Σ^* such that t is in $\mathsf{tuples}(\sigma)$ and $\mathsf{tuples}(\sigma(sch(\Gamma))) \subseteq Sat(\Gamma)$.*

We denote by $\mathsf{Cons}(\Gamma, \mathcal{T})$ the set of all tuples consistent with respect to Γ. □

We illustrate this definition using Example 1. For the condition $\Gamma = (Sal = s')$ we have $Sat(\Gamma) = \{s'\}$. Thus, $es'd$ is not in $\mathsf{Cons}(\Gamma, \mathcal{T})$ because $(e)(ss')(d)$ is

Algorithm 2 Consistent answer

Input: A query Q : `select` X `from` T `[where` Γ`]` and Σ^*
Output: $\mathsf{C\_ans}(Q)$

```
1: C_ans(Q) := ∅ ; ToRem_X := ∅
2: for all σ in Σ* such that X ⊆ sch(σ) do
3:    if |tuples(σ(X))| > 1 then
4:       ToRem_X := ToRem_X ∪ tuples(σ(X))
5:    else
6:       Let x denote the unique tuple in tuples(σ(X))
7:       if sch(Γ) ⊆ sch(σ) and tuples(σ(sch(Γ))) ⊆ Sat(Γ) then
8:          C_ans(Q) := C_ans(Q) ∪ {x}
9: C_ans(Q) := C_ans(Q) \ ToRem_X
10: return C_ans(Q)
```

in Σ^* and $(ss') \not\subseteq Sat(\Gamma)$. On the other hand, for $\Gamma' = (Sal = s) \vee (Sal = s')$ we have $Sat(\Gamma') = \{s, s'\}$. Since $(ss') \subseteq Sat(\Gamma')$, $es'd$ is in $\mathsf{Cons}(\Gamma', \mathcal{T})$. As $es'd$ has been shown to be in $\mathsf{Inc}(\mathcal{T})$, this means that tuples in $\mathsf{Cons}(\Gamma', \mathcal{T})$ may be inconsistent. As will be seen shortly, this implies that inconsistent tuples may participate in the computation of consistent answers.

3.2 Consistent Answers

In what follows, given a schema X, we denote by $\mathsf{True}(X)$, $\mathsf{Cons}(X)$ and $\mathsf{Inc}(X)$ the set of all true, consistent and inconsistent tuples of $\mathcal{T}(X)$, respectively. The consistent answer to a query Q is defined as follows.

Definition 4. *Let T be a table over universe U and FD the set of associated functional dependencies. Given a query Q :* `select` X `from` T `[where` Γ`]`, *the* consistent answer to Q, *denoted* $\mathsf{C\_ans}(Q)$, *is the set of tuples x such that*

1. x is in $\mathsf{Cons}(X)$, and
2. there exists t such that $sch(t) \subseteq sch(Q)$), $t.X = x$ and t is in $\mathsf{Cons}(\Gamma, \mathcal{T})$. □

We point out that when T is consistent, our m-Chase algorithm coincides with the standard Chase algorithm. Thus, the m-tuples in Σ^* are 'isomorphic' to tuples in the chased table, which implies that $\mathsf{C\_ans}(Q)$ is equal to the standard answer as defined in [19].
Based on Definition 4, it can be shown that Algorithm 2 correctly computes the consistent answer to a query Q. It is also easy to see that this algorithm is linear in the number of m-tuples in Σ^*, that is linear in the number of tuples in T.

Example 2. We illustrate Definition 4 and Algorithm 2 in the context of Example 1 where $\Sigma^* = \{(e)(ss')(d), (e')(s')\}$. Denoting $(e)(ss')(d)$ by σ_1 and $(e')(s')$ by σ_2, we have the following:

• For Q_1 : `select` Sal `from` T, the expected answer is $\emptyset$ because σ_1 implies that s and s' are inconsistent. Algorithm 2 runs as follows:
– With σ_1, the test line 3 succeeds. Thus, $ToRem\_X$ is set to $\{s, s'\}$ line 4,

and so, $\mathsf{C\_ans}(Q)$ remains empty.
− With σ_2, the test line 3 fails and the test on line 7 succeeds (since the query involves no selection condition). Thus, $ToRem\_X$ is not changed, and s' is inserted in $\mathsf{C\_ans}(Q)$ line 8. Then, when processing line 9, $\mathsf{C\_ans}(Q)$ is set to $(\{s'\} \setminus \{s, s'\})$, that is to $\emptyset$, as expected.

• For Q_2 : `select` Emp, Sal `from` T, $\mathsf{C\_ans}(Q_2) = \{e's'\}$ as $e's'$ is the only tuple in $\mathsf{Cons}(Emp\ Sal)$. Algorithm 2 runs as follows:
− With σ_1, the test line 3 succeeds, and so es and es' are inserted in $ToRem\_X$ line 4, while $\mathsf{C\_ans}(Q)$ remains empty.
− With σ_2, the test line 3 fails and the test line 7 succeeds. Thus, $ToRem\_X$ is not changed, and $e's'$ is inserted in $\mathsf{C\_ans}(Q)$ line 8. Since $ToRem\_X = \{es, es'\}$, $\mathsf{C\_ans}(Q)$ is not changed on line 9, and we have: $\mathsf{C\_ans}(Q_2) = \{e's'\}$.

• For Q_3 : `select` Emp `from` T `where` $Sal = s'$, we have $Sat(\Gamma) = \{s'\}$ and so, es' is not in $\mathsf{Cons}(\Gamma, \mathcal{T})$. Thus, e is not in $\mathsf{C\_ans}(Q_3)$. Since e' is in $\mathsf{Cons}(\mathcal{T})$ and $e's'$ is in $\mathsf{Cons}(\Gamma, \mathcal{T})$, $\mathsf{C\_ans}(Q_3) = \{e'\}$. Algorithm 2 runs as follows:
− With σ_1, the tests line 3 and line 7 fail (since $Sat(\Gamma) = \{s'\}$ and $\{s, s'\} \not\subseteq \{s'\}$). Thus, $ToRem\_X$ and $\mathsf{C\_ans}(Q)$ remain empty.
− With σ_2, the test line 3 fails and the test on line 7 succeeds. Thus, $ToRem\_X$ is not changed, and e' is inserted in $\mathsf{C\_ans}(Q)$ line 8. Since $\mathsf{C\_ans}(Q)$ is not changed line 9, we have: $\mathsf{C\_ans}(Q_3) = \{e'\}$. □

4 Explanations

Explanations come in response to a user's request either about the status of a given tuple, or about the presence or the absence of a tuple in the consistent answer to a query Q. Typically, these explanations come as a sentence referring to Σ^* possibly followed by tuples or m-tuples. Obviously, such explanations are meaningful to the user, only if she/he has a clear understanding of how the status of a tuple is obtained.

More precisely, we consider two scenarios: In the first, given a tuple t along with its status, the user seeks for explanations about this status. In Example 1, explaining why the tuple es' is true can be done by displaying "es' is true because es' occurs in Σ^*, in m-tuple $(e)(ss')(d)$"; and explaining why es' is inconsistent can be done by displaying "es' is inconsistent because es' occurs in Σ^* but violates $Emp \rightarrow Sal$, as shown in $(e)(ss')(d)$."

In the second scenario, given a query Q, the user seeks for explanations about the presence or absence of a tuple in the answer. In Example 2, explaining why e' is in the answer to Q_3 can be done by displaying: "e' occurs in Σ^* and violates no functional dependencies" (to explain that e' is consistent); and then "e' occurs in the m-tuple $(e')(s')$" of Σ^* that satisfies the selection condition $Sal = s'$.

In order to provide users with intuitive explanations, we propose first the following generic explanations that allow to deal with the first scenario and based on which explanations in the second scenario are generated.

(A) To explain that t is in $\mathsf{True}(\mathcal{T})$, we display "$t$ occurs in Σ^*", followed by an m-tuple σ such that $t \in \mathsf{tuples}(\sigma(sch(t)))$.

(B) To explain that t is in $\mathsf{False}(\mathcal{T})$, we display "$t$ does not occur in Σ^*" meaning that for every σ in Σ^*, $t \notin \mathsf{tuples}(\sigma(sch(t)))$. Providing evidence of this statement is impossible, unless displaying Σ^*, which is not a reasonable option.

(C) To explain that t is in $\mathsf{Inc}(\mathcal{T})$, we display "$t$ occurs in Σ^* but violates functional dependencies", followed by an m-tuple σ such that $t \in \mathsf{tuples}(\sigma(sch(t)))$, $|\mathsf{tuples}(\sigma(sch(t)))| > 1$ and (at least) one functional dependency $X \rightarrow A$ such that $XA \subseteq sch(\sigma)$, $A \in sch(t)$ and $|\mathsf{tuples}(\sigma(A))| > 1$.

(D) To explain that t is in $\mathsf{Cons}(\mathcal{T})$, we display "$t$ occurs in Σ^* and violates no functional dependencies." Providing evidence of this statement is impossible, unless displaying all σ in Σ^* such that t is a sub-tuple of a tuple q in $\mathsf{tuples}(\sigma)$ to show that they all satisfy that $\mathsf{tuples}(\sigma(sch(t))) = \{t\}$. This looks unrealistic as this would generate an important number of m-tuples!

(E) Given a selection condition Γ, explaining that t is in $Sat(\Gamma)$ is straightforwardly done by displaying "t satisfies the selection condition Γ." To explain that t is in $\mathsf{Cons}(\Gamma, \mathcal{T})$ we display "t occurs in an m-tuple in Σ^* that satisfies the selection condition Γ", followed by a σ such that $t \in \mathsf{tuples}(\sigma(sch(t)))$, $sch(\Gamma) \subseteq sch(\sigma)$ and $\sigma(sch(\Gamma))) \subseteq Sat(\Gamma)$.

Now, given Q : `select` X `from` T `[where` Γ`]`, explaining the presence or the absence of a given tuple x in the consistent answer to Q can be done as follows:

- The explanation of the presence of x in the consistent answer is provided by the definition of the query, namely, x is in the answer because x is in $\mathsf{Cons}(\mathcal{T})$ and x has a super-tuple t in $\mathsf{Cons}(\Gamma, \mathcal{T})$. Therefore, if Q involves no selection condition, explanation **(D)** is displayed and, otherwise, explanations **(D)** and **(E)** above are combined and displayed.
- The explanation of the absence of a tuple x from the answer is due to one of the following three reasons:
  (a) x is in $\mathsf{False}(\mathcal{T})$, in which case explanation **(B)** is displayed,
  (b) x is in $\mathsf{Inc}(\mathcal{T})$, in which case explanation **(C)** is displayed,
  (c) x is in $\mathsf{Cons}(\mathcal{T})$ but x has no super-tuple in $\mathsf{Cons}(\Gamma, \mathcal{T})$, in which case explanation **(D)** is displayed followed by: "x has no super-tuple occurring in an m-tuple of T satisfying the selection condition Γ.

Clearly, providing such explanations assumes that the m-Chase table Σ^* has been computed in advance. We also emphasize here that each of the above explanations requires only a single scan of Σ^*.

Indeed, explanations **(B)** and **(D)** only require to know the status of a tuple (which is computed through a scan of Σ^*), whereas explanations **(A)**, **(C)** and **(E)** display an m-tuple from Σ^*, as an evidence of the message. In any case, *all computations necessary for our explanations are polynomial in the size of* T.

We note that explanations as defined above relate to the work of [16, 17], in the sense that our explanations consist in displaying, whenever possible, tuples responsible of the fact being explained. We are currently investigating how our approach could be more formally defined as done in [16, 17], although their approaches do not address explaining inconsistencies as we do.

5 Quality Measures

When querying a possibly inconsistent table, it is important for users to have an idea of the 'amount' of inconsistency in the table. In this section, we aim to provide users with tools and measures that quantify inconsistencies in the table and we do so based on the set of tuples that are true according to the semantics introduced earlier. We require all measures μ to be such that $0 \leq \mu \leq 1$ and $\mu = 1$ when T is consistent, that is when $\mathsf{Inc}(\mathcal{T}) = \emptyset$.

We distinguish two cases here: (a) quality of data in a table and (b) quality of data in a consistent query answer. In either case, in the definitions of our measures, we assume that the m-chased table Σ^* is available.

5.1 Quality of Data in a Table

Given a table T, we first define the *quality of* T, denoted by $Qual(T)$, as the ratio of the number of consistent tuples over the number of true tuples in $\mathcal{T}$:

$$Qual(T) = \frac{|\mathsf{Cons}(\mathcal{T})|}{|\mathsf{True}(\mathcal{T})|} = 1 - \frac{|\mathsf{Inc}(\mathcal{T})|}{|\mathsf{True}(\mathcal{T})|}.$$

Notice that if T is consistent then $Qual(T) = 1$. Moreover, if $\mathsf{True}(\mathcal{T}) = \emptyset$, then Σ^* is empty, implying that T and $\mathsf{Inc}(\mathcal{T})$ are empty as well. Thus, we set $Qual(T) = 1$ when $\mathsf{True}(\mathcal{T}) = \emptyset$.

However, computing $Qual(T)$ based on Σ^* and Definition 2 is not trivial because each tuple should be counted only *once*. Since every sub-tuple of a true tuple is also true and every sub-tuple of a consistent tuple is consistent, the computation is not easy. To see this, consider the case of true tuples in $\Sigma^* = \{(a)(b), (a)(b'), (a')(b)\}$. Here $\mathsf{True}(\mathcal{T})$ consists of ab, ab', $a'b$, along with all their sub-tuples. As a occurs in ab and in ab', counting the sub-tuples of each tuple occurring in Σ^* may lead to an incorrect result. Indeed, we have $\mathsf{True}(\mathcal{T}) = \{ab, ab', a'b, a, a', b, b'\}$ but as a occurs in ab and in ab', the count in this example *should be* 7 (and *not* 12).

Counting the consistent tuples raises an additional difficulty, because in order to conclude that a tuple t is consistent, *every* m-tuple σ in which t occurs must be such that $|\mathsf{tuples}(\sigma(sch(t)))| = 1$. This is why, to compute $Qual(T)$, it is better to count the inconsistent tuples, based on the facts that

(a) $\mathsf{Cons}(\mathcal{T}) = \mathsf{True}(\mathcal{T}) \setminus \mathsf{Inc}(\mathcal{T})$ and
(b) $\mathsf{Cons}(\mathcal{T}) \cap \mathsf{Inc}(\mathcal{T}) = \emptyset$,

which imply that $\frac{|\mathsf{Cons}(\mathcal{T})|}{|\mathsf{True}(\mathcal{T})|} = 1 - \frac{|\mathsf{Inc}(\mathcal{T})|}{|\mathsf{True}(\mathcal{T})|}$.

Example 3. In Example 1 where $\Sigma^* = \{(e)(ss')(d), (e')(s')\}$, we have seen that $\mathsf{Inc}(\mathcal{T}) = \{esd, es'd, es, es', sd, s'd, s, s'\}$ and thus we have $|\mathsf{Inc}(\mathcal{T})| = 8$ and counting the tuples in $\mathsf{True}(\mathcal{T})$ amounts to counting the number of distinct sub-tuples of esd, $es'd$ and $e's'$, which yields 13. Therefore, we have $Qual(T) = 1 - \frac{8}{13}$, that is $Qual(T) = 5/13$. □

It turns out that the computation of $Qual(T)$ is *polynomial in the number of tuples in* T (in fact a single scan of Σ^* is sufficient in order to identify 'maximal' true tuples, based on which all tuples of interest are processed), but the computation of $Qual(T)$ is *exponential in the number of attributes in* U (as the status of every sub-tuple of a maximal tuple has to be determined and as every tuple must be counted only once).

We are currently investigating how to efficiently compute this measure. It seems that level-wise techniques borrowed from those in the well-known Apriori algorithm [1] should be relevant in our context. Moreover, investigating how to compute an approximate value for $Qual(T)$ based on sampling is also an issue that we are currently considering, inspired by the work in [11].

Another reliable but easier way of assessing the quality of T is to focus on functional dependencies. To express such a measure, for each $X \to A$ in FD, let $\mathsf{Inc}(X \to A)$ be the set of all true tuples not satisfying $X \to A$, i.e.,

$$\mathsf{Inc}(X \to A) = \{x \in \mathsf{True}(X) \mid (\exists \sigma \in \Sigma^*)(XA \subseteq sch(\sigma) \wedge x \in \mathsf{tuples}(\sigma(X)) \wedge |\mathsf{tuples}(\sigma(A))| > 1)\}.$$

If $\mathsf{True}(X)$ is nonempty, the associated measure is:

$$Qual_X^A(T) = 1 - \frac{|\mathsf{Inc}(X \to A)|}{|\mathsf{True}(X)|}.$$

Notice that the definition of $Qual_X^A(T)$ satisfies the property that if T satisfies the functional dependency $X \to A$, then $Qual_X^A(T) = 1$, because in this case, $\mathsf{Inc}(X \to A) = \emptyset$. Therefore, if T is consistent then $Qual_X^A(T) = 1$.

We emphasize that $|\mathsf{Inc}(X \to A)|$ and $|\mathsf{True}(X)|$ are computed through a simple scan of Σ^*, which implies that $Qual_X^A(T)$ is computed efficiently. We then define the following measure as the average of all $Qual_X^A(T)$:

$$Qual_{FD}(T) = \frac{\sum_{X \to A \in FD} Qual_X^A(T)}{|FD|}.$$

In Example 3, we have $\mathsf{Inc}(Emp \to Sal) = \{e\}$ and $\mathsf{True}(Emp) = \{e, e'\}$. Hence, $|\mathsf{Inc}(Emp \to Sal)| = 1$ and $|\mathsf{True}(Emp)| = 2$, and so, $Qual_{Emp}^{Sal}(T) = 0.5$. As $|FD| = 1$, we have $Qual_{FD}(T) = 0.5$.

We note that, if T is consistent then $Qual_X^A(T) = 1$, for every $X \to A$ in FD, and therefore $Qual_{FD}(T) = 1$. We also note that the knowledge of $Qual_X^A(T)$ for every $X \to A$ in FD allows for the definition of additional aggregate measures by replacing 'average' by for example 'maximum' or 'minimum'.

Moreover, as in the work in [11] about *approximate* functional dependencies, the use of a threshold ρ such that $0 < \rho \leq 1$, allows to define the measure:

$$Qual_{FD}^{\rho}(T) = \frac{|\{X \to A \in FD \mid Qual_X^A(T) \geq \rho\}|}{|FD|}.$$

Intuitively, ρ is a threshold below which a functional dependency is considered *not* satisfied, and $Qual_{FD}^{\rho}$ returns the ratio of functional dependencies in FD

that are 'approximately satisfied' with respect to ρ. In Example 3, for $\rho = 0.33$, $Qual^{\rho}_{FD} = 1$, because $Emp \rightarrow Sal$ is considered 'approximately satisfied'.

We point out that the knowledge of $Qual_{FD}(T)$ and of every $Qual^{A}_{X}(T)$ allow to put forward the inconsistencies in T. Indeed, if we use an appropriate graphical interface to display all values of $Qual^{A}_{X}(T)$, for every $X \rightarrow A$, then one can easily identify the values of $Qual^{A}_{X}(T)$ considered too low; and then, for each of the corresponding functional dependencies, one can justify the consistency or inconsistency of tuples in $\mathcal{T}(XA)$ using explanations as seen in Section 4.

5.2 Quality of Data in a Consistent Answer

Given Q : `select` X `from` T `[where` Γ`]`, we denote by $\mathsf{Ans}(Q)$ the set of all true tuples x over X having a true super-tuple in $Sat(\Gamma)$. Formally,

$\mathsf{Ans}(Q) = \{x \in \mathsf{True}(X) \mid (\exists q \in \mathsf{True}(sch(Q)))(q.X = x \wedge q.sch(\Gamma) \in Sat(\Gamma))\}$.

Notice that, if T is consistent, then $\mathsf{Ans}(Q) = \mathsf{C\_ans}(Q)$. However, if T is not consistent, there may be tuples in $\mathsf{Ans}(Q)$ which are *not* in $\mathsf{C\_ans}(Q)$.

In the context of Example 2, for the query Q_3, we have $\mathsf{C\_ans}(Q_3) = \{e'\}$, whereas $\mathsf{Ans}(Q_3) = \{e, e'\}$ (because es' and $e's'$ are true and satisfy the condition $(Sal = s')$). However, in all cases, we have $\mathsf{C\_ans}(Q) \subseteq \mathsf{Ans}(Q)$.

We define the quality of the consistent answer to Q as the ratio between the number of tuples in $\mathsf{C\_ans}(Q)$ over the number of tuples in $\mathsf{Ans}(Q)$. Formally, assuming that $\mathsf{Ans}(Q)$ is nonempty:

$$Qual(Q) = \frac{|\mathsf{C\_ans}(Q)|}{|\mathsf{Ans}(Q)|}.$$

If $\mathsf{Ans}(Q) = \emptyset$ then we set $Qual(Q) = 1$ because in this case $\mathsf{C\_ans}(Q)$ is empty. Clearly, if T is consistent then, as $\mathsf{Ans}(Q) = \mathsf{C\_ans}(Q)$, we have $Qual(Q) = 1$. Notice also that if Q involves no selection condition then $Qual(Q) = \frac{|\mathsf{Cons}(X)|}{|\mathsf{True}(X)|}$.

To illustrate this quality measure, refer to Example 2, where the consistent answers to Q_1 and Q_3 are respectively $\{ed\}$ and $\{e'\}$. As we also have $\mathsf{Ans}(Q_1) = \{ed\}$ and $\mathsf{Ans}(Q_3) = \{e, e'\}$, we obtain $Qual(Q_1) = 1$ and $Qual(Q_3) = 0.5$.

It is important to note that, if the contents of query answers are displayed together with their qualities, the user has the opportunity to ask for explanations as described in the previous section. For instance, in our example above, a user might wonder why the quality of the answer to query Q_3 is 0.5. The explanation to be provided in this case is that there exist tuples in $\mathsf{Ans}(Q_3)$ which are not in $\mathsf{C\_ans}(Q_3)$. The user can then continue the dialogue with the system, for example by asking to see such tuples.

6 Conclusion

In this paper, we have addressed two issues regarding query processing in tables with nulls and functional dependencies, namely explanation of a query answer

and quality of data in a query answer. The starting point was our earlier work on defining consistent answers to queries submitted to such tables [13]. Based on that earlier work we have proposed an approach to explaining the content of a consistent query answer and measuring its quality. We have shown that explaining the contents consists mainly in determining why a given tuple is consistent/inconsistent, or why it is or it is not in the consistent answer to a query. We have also proposed measures for assessing the quality of the data in a table and in a query answer, as well as algorithms for their computation. Finally, we have seen how the simultaneous use of explanations and quality measures can help users to better 'understand' inconsistencies in the input table.

We are currently pursuing three lines of research. First, we aim to improve the algorithm for computing $Qual(T)$, because even if the computation is polynomial in the size of T, it is nevertheless exponential in the size of the universe. Second, our approach to explanation needs to be further refined, probably along the lines of [4, 6, 16, 17]; also, investigating interactions between explanations and quality measures is a challenging issue. Third, we are exploring the possible use of sampling techniques for providing approximate measures and further improving performance. Indeed, as our approach is meant to apply to large tables resulting from the integration of data from several sources, even a simple scan of the m-table might prove expensive.

References

1. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A.I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 309–328. AAAI-MIT Press, 1996.
2. Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In Victor Vianu and Christos H. Papadimitriou, editors, *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Pennsylvania, USA*, pages 68–79. ACM Press, 1999.
3. Leopoldo E. Bertossi. *Database Repairing and Consistent Query Answering*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.
4. Leopoldo E. Bertossi. Specifying and computing causes for query answers in databases via database repairs and repair-programs. *Knowl. Inf. Syst.*, 63(1):199–231, 2021.
5. Leopoldo E. Bertossi and Jan Chomicki. Query answering in inconsistent databases. In Jan Chomicki, Ron van der Meyden, and Gunter Saake, editors, *Logics for Emerging Applications of Databases [outcome of a Dagstuhl seminar]*, pages 43–83. Springer, 2003.
6. Leopoldo E. Bertossi and Babak Salimi. Unifying causality, diagnosis, repairs and view-updates in databases. *CoRR*, abs/1405.4228, 2014.
7. Ronald Fagin, Alberto O. Mendelzon, and Jeffrey D. Ullman. A simplified universal relation assumption and its properties. *ACM Trans. Database Syst.*, 7(3):343–360, 1982.
8. John Grant. Measuring inconsistency in generalized propositional logic. *Logica Universalis*, 14(3):331–356, 2020.

9. Joseph Y. Halpern and Judea Pearl. Causes and explanations: A structural-model approach. part II: explanations. *CoRR*, cs.AI/0208034, 2002.
10. Joseph Y. Halpern and Judea Pearl. Causes and explanations: A structural-model approach — part 1: Causes. *CoRR*, abs/1301.2275, 2013.
11. Jyrki Kivinen and Heikki Mannila. Approximate inference of functional dependencies from relations. *Theor. Comput. Sci.*, 149(1):129–149, 1995.
12. Dominique Laurent and Nicolas Spyratos. Handling inconsistencies in tables with nulls and functional dependencies. *J. Intell. Inf. Syst.*, 59(2):285–317, 2022.
13. Dominique Laurent and Nicolas Spyratos. Consistent query answering without repairs in tables with nulls and functional dependencies. *CoRR*, abs/2301.03668, 2023.
14. Ester Livshits, Benny Kimelfeld, and Sudeepa Roy. Computing optimal repairs for functional dependencies. *ACM Trans. Database Syst.*, 45(1):4:1–4:46, 2020.
15. Alexandra Meliou, Wolfgang Gatterbauer, Joseph Y. Halpern, Christoph Koch, Katherine F. Moore, and Dan Suciu. Causality in databases. *IEEE Data Eng. Bull.*, 33(3):59–67, 2010.
16. Alexandra Meliou, Sudeepa Roy, and Dan Suciu. Causality and explanations in databases. *Proc. VLDB Endow.*, 7(13):1715–1716, 2014.
17. Sudeepa Roy and Dan Suciu. A formal approach to finding explanations for database queries. In Curtis E. Dyreson, Feifei Li, and M. Tamer Özsu, editors, *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 1579–1590. ACM, 2014.
18. Nicolas Spyratos. The partition model: A deductive database model. *ACM Trans. Database Syst.*, 12(1):1–37, 1987.
19. Jeffrey D. Ullman. *Principles of Databases and Knowledge-Base Systems*, volume 1-2. Computer Science Press, 1988.
20. Jef Wijsen. On the consistent rewriting of conjunctive queries under primary key constraints. *Inf. Syst.*, 34(7):578–601, 2009.