Data Quality Problems beyond Consistency and Deduplication

Wenfei Fan[1], Floris Geerts[2], Shuai Ma[3], Nan Tang[4], and Wenyuan Yu[1]

[1] University of Edinburgh, UK,
wenfei@inf.ed.ac.uk,wenyuan.yu@ed.ac.uk
[2] University of Antwerp, Belgium,
floris.geerts@ua.ac.be
[3] SKLSDE Lab, Beihang University, China,
mashuai@buaa.edu.cn
[4] Qatar Computing Research Institute, Qatar,
ntang@qf.org.qa

Abstract. Recent work on data quality has primarily focused on data repairing algorithms for improving data consistency and record matching methods for data deduplication. This paper accentuates several other challenging issues that are essential to developing data cleaning systems, namely, error correction with performance guarantees, unification of data repairing and record matching, relative information completeness, and data currency. We provide an overview of recent advances in the study of these issues, and advocate the need for developing a logical framework for a uniform treatment of these issues.

1 Introduction

Data quality has been a longstanding line of research for decades [20]. It is estimated that dirty data costs US companies alone 600 billion dollars each year [9]. With this comes the need for data cleaning systems to improve data quality, and to add accuracy and value to business processes. As an example, data cleaning tools deliver "an overall business value of more than 600 million GBP" each year at BT [31]. In light of this, the market for data cleaning systems is growing at 17% annually, substantially outpacing the 7% average of other IT segments [21].

There has been a host of work on data quality. Recent work has primarily focused on two central issues:

- *Recording matching*: to identify tuples that refer to the same real-world entity [10], for data deduplication.
- *Data repairing*: to find a repair (database) that is consistent *w.r.t.* integrity constraints and minimally differs from the original data, by detecting and fixing (semantic) errors, to improve data consistency [1].

Most data cleaning systems on the market support record matching, *e.g.,* ETL tools (extraction, transformation, loading; see [24] for a survey). Some prototype systems also provide a data repairing functionality [3, 6, 28, 37].

There are other data quality issues that are not limited to algorithms for record matching or data repairing, but are also essential to developing practical data cleaning systems. Unfortunately, these issues have not received much attention from the research community. In particular, we highlight the following.

(1) Certain fixes. Prior data repairing methods are typically heuristic. They attempt to fix all the errors in the data, but do not guarantee that the generated fixes are correct. Worse still, new errors may be introduced when trying to repair the data. In practice, we often want to find *certain fixes*, *i.e.,* fixes that are guaranteed to be correct, although we might not be able to fix *all* the errors in the data. The need for certain fixes is particularly evident when repairing critical data, *e.g.,* medical data, in which a seemingly minor error may mean life or death.

(2) Unification of data repairing and record matching. Data repairing and record matching are typically treated as independent processes. However, the two processes often interact with each other: repairing helps us identify matches, and vice versa. This suggests that we unify repairing and matching by interleaving their operations.

(3) Information completeness. A data cleaning system should be able to tell us, given a database D and a query Q, whether D has complete information to answer Q. If the information is missing from D, the answer to Q in D is hardly sensible. Information completeness is as important as data consistency and deduplication. Indeed, pieces of information perceived as being needed for clinical decisions were missing from 13.6% to 81% of the time [29]. Traditionally we deal with this issue by adopting either the Closed World Assumption (CWA) or the Open World Assumption (OWA). However, real-life databases are often neither entirely closed-world nor entirely open-world. This asks for a revision of the CWA, OWA and the model of information completeness.

(4) Data currency. The quality of data in a real-life database quickly degenerates over time. It is estimated that "2% of records in a customer file become obsolete in one month" [9]. That is, in a database of 500 000 customer records, 10 000 records may go stale per month, 120 000 records per year, and within two years about 50% of all the records may be obsolete. As a result, we often find that multiple values of the same entity reside in a database, which were *once correct*, *i.e.,* they were true values

	FN	LN	AC	phn	type	str	city	zip	item	when	where
t_1:	Bob	Brady	020	079172485	2	null	Edi	EH7 4AH	CD	7pm, 28/08/2010	UK
t_2:	Max	Smith	131	6884593	1	5 Oak St	Ldn	EH8 9HL	CD	06/11/2009	UK
t_3:	Mark	Smith	131	6884563	1	null	Edi	null	DVD	1pm, 06/11/2009	US

(a) Example input tuples t_1 and t_2

	FN	LN	AC	Hphn	Mphn	str	city	zip	gender
s_1:	Robert	Brady	131	6682845	079172485	51 Elm Row	Edi	EH7 4AH	M
s_2:	Mark	Smith	131	6884563	075568485	5 Oak St	Edi	EH8 9HL	M

(b) Example master relation D_m

Fig. 1. Example input tuples and master relation

of the entity at some time, but most of them have become *obsolete* and *inaccurate*. This highlights the need for studying *data currency*, to identify the current values of entities in a database, and to answer queries with the current values.

This paper aims to bring attention to these issues. We present an overview of recent work on these four issues (in Sections 2– 5, respectively). We argue that these issues interact with each other and also interact with data repairing and record matching; they should be uniformly treated in a logical framework (Section 6). We refer to the monograph [15] for a more complete treatment of these issues.

2 Certain Fixes instead of Heuristics Repairs

Data repairing detects and fixes errors by using integrity constraints, such that data conflicts and errors emerge as violations of the constraints. A variety of constraints have been studied for data repairing, such as denial constraints [3], traditional functional and inclusion dependencies [1], and conditional dependencies [4,6,16,37].

Integrity constraints are capable of detecting whether the data is dirty, *i.e.,* the presence of errors in the data. However, they do not tell us which attributes of a tuple have errors and how we should correct the errors.

Example 1: Consider an input tuple t_1 given in Fig. 1(a). It specifies a transaction record (tran) of a credit card: an item purchased at place where and time when, by a UK customer who is identified by name (FN, LN), phone number (area code AC and phone phn) and address (street str, city, zip code). Here phn is either home phone or mobile phone, indicated by type (1 or 2, respectively). It is known that when AC is 020, city should be London (Ldn), and when AC is 131, city must be Edinburgh (Edi). This semantics of the data can be expressed as conditional functional dependencies (CFDs [16]). The CFDs detect that tuple t_1 is *inconsistent*:

$t_1[\mathsf{AC}] = 020$ but $t_1[\mathsf{city}] =$ Edi. However, they do not tell us which of $t_1[\mathsf{AC}]$ and $t_1[\mathsf{city}]$ is wrong, and to what value it should be changed.

In light of this, prior data repairing methods are *heuristic*: they do not guarantee to find correct fixes in data repairing. Worse still, they may introduce new errors when trying to repair the data. Indeed, the correct values of $t_1[\mathsf{AC}, \mathsf{city}]$ are (131, Edi). Nevertheless, all of the prior methods may opt to change $t_1[\mathsf{city}]$ to Ldn; this does not fix the erroneous attribute $t_1[\mathsf{AC}]$ and worse still, messes up the correct attribute $t[\mathsf{city}]$. $\square$

In practice it is often necessary to guarantee each fix to be *certain*, *i.e.,* assured correct (validated). This can done by using master data and editing rules. *Master data* (*a.k.a. reference data*) is a single repository of high-quality data that provides various applications in an enterprise with a synchronized, consistent view of its core business entities [27]. It is increasingly common for enterprises to maintain master data. *Editing rules* tell us which attributes of a tuple are wrong and what values from master data they should take, provided that some attributes are validated. As opposed to integrity constraints, they specify updates and have a *dynamic* semantics.

Example 2: A master relation D_m is shown in Fig. 1(b). Each tuple in D_m specifies a UK credit card holder (card) in terms of the name, home phone (Hphn), mobile phone (Mphn), address and gender. Consider the following editing rules:

– eR_1: for an input tuple t, if there exists a master tuple s in D_m such that $s[\mathsf{zip}] = t[\mathsf{zip}]$, then t should be updated by $t[\mathsf{AC}, \mathsf{str}, \mathsf{city}] := s[\mathsf{AC}, \mathsf{str}, \mathsf{city}]$, provided that $t[\mathsf{zip}]$ is validated (*e.g.,* assured by the users).
– eR_2: if $t[\mathsf{type}] = 2$ (indicating mobile phone) and if there exists a master tuple s with $s[\mathsf{phn}] = t[\mathsf{Mphn}]$, then $t[\mathsf{FN}, \mathsf{LN}] := s[\mathsf{FN}, \mathsf{LN}]$, as long as $t[\mathsf{phn}, \mathsf{type}]$ are already validated.

When $t_1[\mathsf{zip}]$ is assured correct, eR_1 *corrects* attribute $t_1[\mathsf{AC}]$ and enriches $t_1[\mathsf{str}]$ by taking values from master data $s_1[\mathsf{AC}, \mathsf{str}]$. Note that when the editing rule and $t_1[\mathsf{zip}]$ are validated, the fix to $t_1[\mathsf{AC}]$ is certainly correct. Similarly, when $t_1[\mathsf{Mphn}, \mathsf{type}]$ are validated, eR_2 *standardizes* $t_1[\mathsf{FN}]$ by changing Bob to Robert. $\square$

Certain fixes. More specifically, we define certain fixes as follows (see [19] for details). Consider an input tuple t and a set Z of attributes such that $t[Z]$ is validated. We use $t \rightarrow_{(\varphi, t_m, Z)} t'$ to denote that tuple t' is obtained from t by means of updates specified in an editing rule φ with a master tuple t_m. We denote by $\mathsf{ext}(Z, \varphi, t_m)$ the *validated region* of t', which includes attributes in Z and the attributes updated by φ with t_m.

Given a set Θ of editing rules and master data D_m, we say that a tuple t' is a *fix* of t by (Θ, D_m), denoted by $t \rightarrow^*_{(\Theta, D_m, Z)} t'$, if there exists a finite sequence $t_0 = t$, t_1, …, $t_k = t'$ of tuples, and for each $i \in [1, k]$, there exists an editing rule $\varphi_i \in \Theta$ and a master tuple $t_{m_i} \in D_m$ such that (a) $t_{i-1} \rightarrow_{(\varphi_i, t_{m_i}, Z_{i-1})} t_i$, where $Z_i = \mathsf{ext}(Z_{i-1}, \varphi_i, t_{m_{i-1}})$; (b) $t_i[Z] = t[Z]$; and (c) for all $\varphi \in \Theta$ and $t_m \in D_m$, $t' \not\rightarrow_{(\varphi, t_m, Z_m)} t'$. Intuitively, (a) each step of the correcting process is justified; (b) $t[Z]$ is validated and hence, remains unchanged; and (c) t' is a fixpoint and cannot be further updated, *i.e.,* the changes incurred to t by (Θ, D_m) are "maximum".

We say that t has a *certain fix* by (Θ, D_m) *w.r.t.* Z if there exists a *unique* t' such that $t \rightarrow^*_{(\Theta, D_m, Z)} t'$.

Given a set Θ of editing rules and master data D_m, one can monitor input tuples and find their certain fixes. For each tuple t, the user may assure that a (possible empty) set $t[Z]$ of attributes is correct. There is an algorithm that, given Z, iteratively employs Θ and D_m to find a certain fix for as many attributes in t as possible. The correctness of the fix is guaranteed by master data and editing rules. As opposed to data repairing, we do not stress fixing all the attributes of t by requiring the users to validate a large region $t[Z]$. Nevertheless, when the users opt to find a certain fix for the entire t, there is an algorithm that, given Z, identifies a *minimal* set Z' of attributes such that when $t[Z \cup Z']$ is validated, a certain fix for t is warranted [19]. One can recommend $t[Z']$ to the users for validating, and the users may respond with more validated attributes (not necessarily $t[Z']$). From these an interactive process readily follows that proceeds until all the attributes of t are validated.

Fundamental problems. There are several important problems associated with certain fixes. Consider tuples of a relation schema R. One problem is to determine, given a set Θ of editing rules, master data D_m, and a set Z of attributes of schema R, whether for all tuples t of R, if $t[Z]$ is validated then t has a certain fix by (Θ, D_m). In other words, it is to determine whether Θ and D_m have conflicts. Another problem is to find, given Θ and D_m, a minimal set Z of attributes such that for all tuples t of schema R, if $t[Z]$ is validated then all the attributes of t can be validated by (Θ, D_m). Intuitively, it is to find a minimal region for the users to validate. It is shown that these are intractable [19], but efficient heuristic algorithms have been developed for these problems.

3 Interaction between Repairing and Record Matching

Current data cleaning systems typically treat data repairing and record matching as separate processes, executed consecutively one after another. In practice, the two processes often interact with each other, as illustrated below.

Example 3: Consider the transaction records of Fig. 1(a) and master data for credit card holders given in Fig. 1(b), referred to as tran and card tuples, respectively. Following [11,16], we use CFDs [16] φ_1–φ_2 to specify the consistency of the tran data, and a *matching dependency* (MD) [11] ψ as a rule for matching tran records and card tuples:

φ_1: $\mathsf{tran}([\mathsf{AC} = 131] \rightarrow [\mathsf{city} = \text{Edi}])$,
φ_2: $\mathsf{tran}([\mathsf{type} = 1, \mathsf{city}, \mathsf{phn}] \rightarrow [\mathsf{str}, \mathsf{AC}, \mathsf{zip}])$,
ψ: $\mathsf{tran}[\mathsf{LN}, \mathsf{city}, \mathsf{str}, \mathsf{zip}] = \mathsf{card}[\mathsf{LN}, \mathsf{city}, \mathsf{str}, \mathsf{zip}] \wedge \mathsf{tran}[\mathsf{FN}] \approx \mathsf{card}[\mathsf{FN}]$
$\wedge\ \mathsf{tran}[\mathsf{type}] = 1 \rightarrow \mathsf{tran}[\mathsf{FN}, \mathsf{phn}] \rightleftharpoons \mathsf{card}[\mathsf{FN}, \mathsf{Hphn}]$

Here (1) CFD φ_1 asserts that if the area code is 131, the city must be Edi; (2) CFD φ_2 states that when $\mathsf{type} = 1$ (*i.e.,* phn is mobile phone), city and home phone uniquely determine street, area code and zip code; and (3) MD ψ assures that for any tran record t and any card tuple, if they have the same last name and address, and if their first names are *similar*, then their home phone and FN attributes can be identified (when $t[\mathsf{type}] = 1$).

Consider tuples t_2 and t_3 in Fig. 1(a). One suspects that the two refer to the same person. If so, then these records show that the same person made purchases in the UK and in the US at about the same time (taking into account the 5-hour time difference between the two countries), indicating that a fraud has likely been committed.

Observe that t_2 and t_3 are quite different in their FN, city, str, zip and phn attributes. No rule allows us to identify the two directly. Nonetheless, they can be matched by *interleaved* matching and repairing operations:

(a) get a repair t_2' of t_2 such that $t_2'[\mathsf{city}] = \text{Edi}$ by applying CFD φ_1 to t_2;

(b) match t_2' with master tuple s_2, to which MD ψ can be applied; as a result of the matching operation, get a repair t_2'' of t_2 by correcting $t_2''[\mathsf{phn}]$ with the master data $s_2[\mathsf{Hphn}] = 6884563$;

(c) find a repair t_3' of t_3 by applying CFD φ_2 to t_2'' and t_3: since t_2'' and t_3 agree on their city and phn attributes and $t_2''[\mathsf{type}] = t_3[\mathsf{type}] = 1$, φ_2 can be applied. This allows us to enrich $t_3[\mathsf{str}]$ and fix $t_3[\mathsf{zip}]$ by taking corresponding values from t_2'', which have been confirmed correct with the master data in step (b).

Note that t''_2 and t'_3 agree on every attribute in connection with personal information. It is evident that they indeed refer to the same person; hence a fraud. Observe that not only repairing helps matching (*e.g.,* from step (a) to (b)), but matching also helps us repair the data (*e.g.,* step (c) is doable only after the matching in (b)). □

Unification. The example tells us the following. (1) When taken together, record matching and data repairing perform much better than being treated as separate processes. (2) To make practical use of their interaction, matching and repairing operations should be *interleaved*, rather than executing the two processes one after another. Unifying matching and repairing, we state the data cleaning problem as follows.

Given a database D, master data D_m, integrity constraints Σ and matching rules Γ, the *data cleaning problem* is to find a repair D_r of D such that (a) D_r is *consistent* (*i.e.,* satisfying Σ), (b) no more tuples in D_r can be *matched* to master tuples in D_m by matching rules of Γ, and (c) D_r minimally differs from the original data D.

The interaction between repairing and matching has been observed in, *e.g.,* [8, 18, 36]. Here, [8, 36] investigate record matching in the presence of error data, and suggest to integrate matching and data merge/fusion. In [18], a rule-based framework is proposed in which CFDs and MDs are both treated as *cleaning rules*. These rules tell us how to fix errors by updating the data, and allow us to interleave repairing and matching operations. Based on these rules, algorithms have been developed to clean data, in the presence or in the absence of master data. It has been shown that by unifying repairing and matching, these algorithms substantially improve the accuracy of repairing and matching taken as separate processes [18].

Fundamental problems. When integrity constraints (for data repairing) and matching rules (for record matching) are taken together, the classical consistency and implication problems for constraints need to be revisited. These issues are investigated for CFDs and MDs in [18], which shows that these problems remain to be NP-complete and coNP-complete, respectively, the same as their counterparts for CFDs alone.

There are two fundamental questions about rule-based data cleaning. The *termination problem* is to decide whether a cleaning process stops, *i.e.,* it reaches a *fixpoint*, such that no more rules can be applied. The *determinism problem* asks whether all terminating cleaning processes end up with the same repair, *i.e.,* all of them reach a *unique* fixpoint. When

CFDs and MDs are treated as cleaning rules, both problems are PSPACE-complete [18].

4 Relative Information Completeness

Given a database D and a query Q, we want to know whether a complete answer to Q can be found in D. Traditional work on this issue adopts either the CWA or the OWA. The CWA assumes that a database contains all the tuples representing real-world entities, but the *values* of some attributes in those tuples are possibly *missing*. The OWA assumes that *tuples* may also be *missing* [35]. As remarked earlier, few real-life databases are closed-world. Under the OWA, one can often expect few sensible queries to find complete answers.

Databases in real world are often neither entirely closed-world nor entirely open-world. This is particularly evident in the presence of master data. Master data of an enterprise contains complete information about the enterprise in certain aspects, *e.g.,* employees and projects, and can be regarded as a closed-world database. Meanwhile a number of other databases may be in use in the enterprise. On one hand, these databases may not be complete, *e.g.,* some sale transactions may be missing. On the other hand, certain parts of the databases are *constrained by* the master data, *e.g.,* employees. In other words, these databases are *partially closed.*

Example 4: Consider a company that maintains $\mathsf{DCust}(\mathsf{cid}, \mathsf{name}, \mathsf{AC}, \mathsf{phn})$, a master data relation consisting of all its domestic customers, in which a tuple (c, n, a, p) specifies the id c, name n, area code a and phone number p of a customer. In addition, the company also has databases (a) $\mathsf{Cust}(\mathsf{cid}, \mathsf{name}, \mathsf{CC}, \mathsf{AC}, \mathsf{phn})$ of all customers of the company, domestic (with country code $\mathsf{CC} = 01$) or international; and (b) $\mathsf{Supt}(\mathsf{eid}, \mathsf{dept}, \mathsf{cid})$, indicating that employee eid in dept supports customer cid. Neither Cust nor Supt is part of the master data.

Consider query Q_1 posed on Supt to find all the customers in NJ with $\mathsf{AC} = 908$ who are supported by the employee with $\mathsf{eid} = e_0$. The query may *not* get a complete answer since some tuples may be missing from Supt. However, if Q_1 returns all NJ customers with $\mathsf{AC} = 908$ found in master data DCust, then we can safely conclude that Supt is complete for Q_1 and hence, there is no need to add more tuples to Supt to answer Q_1.

Now consider a query Q_2 to find *all* customers supported by e_0. Note that the international customers of Cust are not constrained by master data. As a result, we are not able to tell whether any Supt tuples in connection with e_0 are missing. Worse still, we do not even know what tuples should be added to Supt to make the answer to Q_2 in Supt complete. Nevertheless, if we know that (eid $\rightarrow$ dept, cid) is a functional dependency (FD) on Supt, then we can also conclude that the answer to Q_2 in Supt is complete as long as it is nonempty. $\square$

Relative information completeness. A practical data cleaning system should be able to decide whether a database has complete information to answer a query. To this end, as shown by the example, we need a model to specify partially closed databases. There has been a host of work on incomplete information, notably representation systems (*e.g.,* c-tables, v-tables [23, 25]) and models for missing tuples [22, 26, 30] (see [35] for a survey). However, the prior work neither considers master data nor studies the question mentioned above.

Given a database D and master data D_m, we specify a set V of *containment constraints* [13]. A containment constraint is of the form $q(D) \subseteq p(D_m)$, where q is a query posed on D, and p is a simple projection query on D_m. Intuitively, the part of D that is constrained by V is bounded by D_m, while the rest is open-world. We refer to a database D that satisfies V as a *partially closed* database *w.r.t.* (D_m, V). A database D' is a *partially closed extension* of D if $D \subseteq D'$ and D is partially closed *w.r.t.* (D_m, V) itself.

A partially closed database D is said to be *complete for a query Q relative to* (D_m, V) if for all partially closed extensions D' of D *w.r.t.* (D_m, V), $Q(D') = Q(D)$. That is, there is no need for adding new tuples to D, since they either violate the containment constraints, or do not change the answer to Q. In other words, D already contains complete information necessary for answering Q (see [12, 13] for details).

Fundamental problems. One problem is to determine, given a query Q, master data D_m, a set V of containment constraints, and a partially closed database D *w.r.t.* (D_m, V), whether D is complete for Q relatively to (D_m, V). Another problem is to decide, given Q, D_m and V, whether there exists a partially closed database D that is complete for Q relatively to (D_m, V). The analyses of these problems help us identify what data should be collected in order to answer a query. These problems are investigated in [12, 13]. As indicated by Example 4, the complexity of these problems varies depending on different queries and containment constraints [12,13].

5 Data Currency

A data cleaning system should support data currency analysis: among multiple (possibly obsolete) values of an entity, it is to identify the latest

	FN	LN	country	zip	street	salary	status
t_1:	Mary	Smith	UK	OX1 3QD	2 Small St	50k	single
t_2:	Mary	Dupont	UK	EB21 5FX	10 Elm Ave	50k	married
t_3:	Mary	Dupont	UK	EH9 1SU	6 Main St	80k	married
t_4:	Bob	Luth	UK	DB9 FJ8	8 Cowan St	80k	married

(a) Relation Emp

	dname	mgrFN	mgrLN	mgrAddr	budget
s_1:	R&D	Mary	Smith	2 Small St, OX1 3QD, UK	6500k
s_2:	R&D	Mary	Smith	2 Small St, OX1 3QD, UK	7000k
s_3:	R&D	Mary	Dupont	6 Main St,EH9 1SU, UK	6000k
s_4:	R&D	Ed	Luth	8 Cowan St, DB9 FJ8, UK	6000k

(b) Relation Dept

Fig. 2. A company database

value of the entity, and to answer queries using the latest values only. The question of data currency would be trivial if all data values carried valid timestamps. In practice, however, timestamps are often unavailable or imprecise [38]. Add to this the complication that data values are often copied or imported from other sources [2, 7], which may not support a uniform scheme of timestamps.

Not all is lost. It is often possible to deduce currency orders from the semantics of the data. Moreover, data copied from other sources inherit currency orders in those sources. Taken together, these often allow us to deduce sufficient current values of the data to answer certain queries, as illustrated below.

Example 5: Consider two relations of a company shown in Fig. 2. Each Emp tuple is an employee record with name, address (country, zip code, street), salary and marital status. A Dept tuple specifies the name, manager and budget of a department. Records in these relations may be stale, and do not carry timestamps. Here tuples t_1, t_2 and t_3 refer to the same employee Mary, while t_4 does not refer to Mary. Consider the following queries posed on these relations.

(1) Query Q_1 is to find Mary's current salary. No timestamps are available for us to tell which of 50k or 80k is more current. However, we may know that the salary of each employee in the company does *not* decrease, as commonly found in the real world. This yields currency orders $t_1 \prec_{\mathsf{salary}} t_3$ and $t_2 \prec_{\mathsf{salary}} t_3$, *i.e.,* t_3 is *more current* than t_1 and t_2 in *attribute* salary; in other words, $t_3[\mathsf{salary}]$ is more current than both $t_1[\mathsf{salary}]$ and $t_2[\mathsf{salary}]$. Hence the answer to Q_1 is 80k.

(2) Query Q_2 is to find Mary's current last name. We can no longer answer Q_2 as above. Nonetheless, we may know the following: (a) marital status can only change from single to married and from married to divorced;

but not from married to single; and (b) Emp tuples with the most current marital status also contain the most current last name. Therefore, $t_1 \prec_{\mathsf{LN}} t_2$ and $t_1 \prec_{\mathsf{LN}} t_3$, and the answer to Q_2 is Dupont.

(3) Query Q_3 is to find Mary's current address. We may know that Emp tuples with the most current salary contain the most current address. From this and (1) above, we know that the answer to Q_3 is "6 Main St".

(4) Finally, query Q_4 is to find the current budget of department R&D. Again no timestamps are available for us to evaluate the query. However, we may know the following: (a) Dept tuples s_1 and s_2 have copied their mgrAddr values from t_1[street, zip, county] in Emp; similarly, s_3 has copied from t_3, and s_4 from t_4; and (b) in Dept, tuples with the most current address also have the most current budget. Taken together, these tell us that $s_1 \prec_{\mathsf{budget}} s_3$ and $s_2 \prec_{\mathsf{budget}} s_3$. Observe that we do not know which budget in s_3 or s_4 is more current. Nevertheless, in either case the most current budget is 6000k, and hence it is the answer to Q_4. □

Modeling data currency. To study data currency we need to specify currency orders on data values in the absence of timestamps but in the presence of copy relationships. Such a model is recently proposed in [17].

(1) To model partially available currency information in a database D, it assumes a currency order $\prec_A$ for each attribute A, such that for tuples t_1 and t_2 in D that represent the same real-world entity, $t_1 \prec_A t_2$ indicates that t_2 is more up-to-date than t_1 in the A attribute value.

(2) It uses denial constraints [1] to express currency relationships derived from the semantics of the data. For instance, all the currency relations we have seen in Example 5 can be expressed as denial constraints.

(3) A copy function from a data source to another is defined in terms of a partial mapping that preserves the currency order in the source. Based on these, one can define *consistent completions* D^c of D, which extend $\prec_A$ in D to a total order on all tuples pertaining to the same entity, such that D^c satisfies the denial constraints and constraints imposed by the copy functions.

One can construct from D^c the *current tuple* for each entity *w.r.t.* $\prec_A$, which contains the entity's most current A value for each attribute A. This yields the *current instance* of D^c consisting of only the current tuples of the entities in D, from which currency orders can be removed. In light of this, one can compute *certain current answers* of a query Q in D, *i.e.,* tuples that are the answers to Q in *all* consistent completions D^c of D (see [17] for details).

	FN	LN	country	zip	street	phone	grade	salary	status
r_1:	Mary	Smith	UK	OX1 3QD	2 Small St	66757574	10	50k	single
r_2:	Mary	Dupont	UK	EB21 5FX	10 Elm Ave	66757574	10	50k	married
r_3:	Mary	Dupont	UK	EH9 1SU	6 Main St	66757574	11	80k	married
r_4:	Bob	Luth	UK	DB9 FJ8	8 Cowan St	46357642	11	80k	married
r_5:	Bob	Luth	UK	DB9 FJ8	8 Cowan St	46357642	12	100k	married

Fig. 3. Relation EmpHistory

The study of data currency is related to temporal databases, which assume the availability of timestamps (see [32] for a survey). Also related is the line of work on querying indefinite data (see, *e.g.,* [34]), which considers data that is linearly ordered but only provides a partial order, but does not evaluate queries using current instances. Algorithms for discovering copy dependencies and functions are developed in [2, 7].

Fundamental problems. Given a database D on which partial currency orders, denial constraints and copy functions $\overline{\rho}$ are defined, we want to determine (1) whether a value is more up-to-date than another, and (2) whether a tuple is a certain current answer to a query. In addition, about copy functions $\overline{\rho}$, we want to determine (3) whether $\overline{\rho}$ is *currency preserving* for a query Q, *i.e.,* no matter how we extend $\overline{\rho}$ by copying more values of those entities in D, the certain current answers to Q in D remain unchanged; and (4) whether $\overline{\rho}$ can be extended to be currency preserving for Q. These problems have been studied in [17] for different queries.

6 Open Research Issues

It is evident that functionalities for handling these issues should logically become part of a data cleaning system. We envisage that a data cleaning system should be able not only to detect data inconsistencies and duplicates, but it should also be able to compute certain fixes that are guaranteed correct. Moreover, it should also be able to improve data currency and information completeness, beyond data consistency and deduplication. Indeed, we naturally want data quality management to tell us whether the answers to our queries in a database are trustable or not. This requires that we take data consistency, currency and information completeness together, as illustrated in the example below.

Example 6: Consider the relation Emp shown in Fig. 2, and a master relation EmpHistory consisting of *all* the historical information of its employees, as shown in Fig. 3. Each EmpHistory tuple is an employee record with name, address (country, zip code, street), phone, grade, salary and marital status. Two constant CFDs are posed on relations Emp and EmpHistory: φ_1 : Emp([country = UK, zip = "EH9 1SU"] $\rightarrow$ [street = "6 Main St"]), and φ_2 : Emp([country =UK, zip = "DB9 FJ8"] $\rightarrow$ [street = "8 Crown St"]), where φ_1 states that in the UK, if one's zip code is "EH9 1SU", its street should be "6 Main St"; similarly, φ_2 states that in the UK, if one's zip code is "DB9 FJ8", its street should be "8 Crown St".

(1) Query Q_1 is to find Mary's current salary. Recall from Example 5(1) that Mary's most current salary is derived to be 80k, drawn from tuple t_3 in relation Emp. Observe the following: (a) t_3 is consistent as it satisfies the CFDs. (b) Mary's salary information gathered in relation Emp is complete *w.r.t.* EmpHistory, since it contains all Mary's employment records in EmpHistory table. Hence we can trust that the answer to Q_1 is 80k, since the data is consistent and the information about Mary is complete.

(2) Query Q_2 is to find Bob's current salary. The only record about Bob in Emp is t_4. Note that t_4 is consistent since it satisfies the CFDs. The answer to Q_2 is 55K in Emp. However, the information about Bob is not complete: there are more records about Bob in EmpHistory, with higher salaries. In other words, relation Emp alone is not sufficient to answer Q_2 correctly. Hence we cannot trust 55K to be the answer to Q_2.

This example demonstrates that to determine whether our queries can be answered correctly, all of data consistency, data currency and information completeness have to be taken into account. □

No matter how important, however, we are not aware of any data cleaning system that supports functionalities to handle all these central data quality issues. The study of these issues is still in its infancy, and it has raised as many questions as it has answered. Below we highlight some of the open issues.

Certain fixes. One question is how to find certain fixes in the absence of master data. Another question concerns methods for discovering editing rules. Indeed, it is unrealistic to rely solely on human experts to design editing rules via an expensive and long manual process. It is likely, however, that editing rules can be deduced from master data and constraints such as CFDs and MDs, for which discovery algorithms are already in place [5, 33].

Relative information completeness and data currency. While the fundamental problems for these issues have been studied, efficient algorithms have yet to be developed and incorporated into data cleaning systems.

A uniform logical framework. To answer a query using a database D, one naturally wants D to be both complete and consistent for Q, and moreover, does not contain duplicates and stale data. In addition, there are intimate connections between these issues. (1) Improving data completeness provides us with more information to repair and match the data, and conversely, data repairing and record matching help us enrich the data as shown in Example 2. (2) Identifying the current value of an entity helps resolve data inconsistencies and duplication, and repairing and matching help us remove obsolete data. (3) Data currency is essentially to deal with missing temporal information, and hence can naturally capitalize on techniques for relative information completeness such as containment constraints and master data. All these highlight the need for developing a uniform framework to handle certain fixes, data repairing, record matching, relative information completeness and data currency. The framework should support the interaction of these processes, to improve the accuracy of data cleaning.

It is both natural and feasible to develop such a framework based on constraints and master data (see *e.g.,* [14] for a initial attempt in this direction). Indeed, (1) constraints are typically used to capture inconsistencies (*e.g.,* [1, 4, 16]). (2) Record matching rules [11] and editing rules [19] can be expressed as dynamic constraints. (3) It is shown [13] that constraints for data consistency, such as denial constraints [1] and conditional dependencies [4, 16], are expressible as simple containment constraints studied for relative information completeness. As a result, we can assure that only consistent and partially closed databases are considered by enforcing containment constraints. (4) It suffices to express data currency commonly found in practice as denial constraints [17], the same class of constraints for data consistency. (5) As remarked earlier, master data has proved effective in dealing with each and every of these issues.

References

1. M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. *TPLP*, 3(4-5):393–424, 2003.
2. L. Berti-Equille, A. D. Sarma, X. Dong, A. Marian, and D. Srivastava. Sailing the information ocean with awareness of currents: Discovery and application of source dependence. In *CIDR*, 2009.
3. P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, 2005.
4. L. Bravo, W. Fan, and S. Ma. Extending dependencies with conditions. In *VLDB*, 2007.
5. F. Chiang and R. Miller. Discovering data quality rules. *PVLDB*, 1(1), 2008.

6. G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving data quality: Consistency and accuracy. In *VLDB*, 2007.
7. X. Dong, L. Berti-Equille, and D. Srivastava. Truth discovery and copying detection in a dynamic world. In *VLDB*, 2009.
8. X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD*, 2005.
9. W. W. Eckerson. Data quality and the bottom line: Achieving business success through a commitment to high quality data. The Data Warehousing Institute, 2002.
10. A. Elmagarmid, P. Ipeirotis, and V. Verykios. Duplicate record detection: A survey. *TKDE*, 19(1):1–16, 2007.
11. W. Fan, H. Gao, X. Jia, J. Li, and S. Ma. Dynamic constraints for record matching. *VLDB J.*, 20(4):495–520, 2011.
12. W. Fan and F. Geerts. Capturing missing tuples and missing values. In *PODS*, 2010.
13. W. Fan and F. Geerts. Relative information completeness. *TODS*, 35(4), 2010.
14. W. Fan and F. Geerts. Uniform dependency language for improving data quality. *IEEE Data Eng. Bull.*, 34(3):34–42, 2011.
15. W. Fan and F. Geerts. *Foundations of Data Quality Management.* Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.
16. W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *TODS*, 33(2), 2008.
17. W. Fan, F. Geerts, and J. Wijsen. Determining the currency of data. In *PODS*, 2011.
18. W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Interaction between record matching and data repairing. In *SIGMOD*, 2011.
19. W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Towards certain fixes with editing rules and master data. *VLDB J.*, 21(2):213–238, 2012.
20. I. Fellegi and D. Holt. A systematic approach to automatic edit and imputation. *J. American Statistical Association*, 71(353):17–35, 1976.
21. Gartner. Forecast: Data quality tools, worldwide, 2006-2011. Technical report, Gartner, 2007.
22. G. Gottlob and R. Zicari. Closed world databases opened through null values. In *VLDB*, 1988.
23. G. Grahne. *The Problem of Incomplete Information in Relational Databases.* Springer, 1991.
24. T. N. Herzog, F. J. Scheuren, and W. E. Winkler. *Data Quality and Record Linkage Techniques.* Springer, 2009.
25. T. Imieliński and W. Lipski, Jr. Incomplete information in relational databases. *JACM*, 31(4), 1984.
26. A. Y. Levy. Obtaining complete answers from incomplete databases. In *VLDB*, 1996.
27. D. Loshin. *Master Data Management.* Knowledge Integrity, Inc., 2009.
28. C. Mayfield, J. Neville, and S. Prabhakar. ERACER: a database approach for statistical inference and data cleaning. In *SIGMOD*, 2010.
29. D. W. Miller et al. Missing prenatal records at a birth center: A communication problem quantified. In *AMIA Annu Symp Proc.*, 2005.
30. A. Motro. Integrity = validity + completeness. *TODS*, 14(4), 1989.
31. B. Otto and K. Weber. From health checks to the seven sisters: The data quality journey at BT, Sept. 2009. BT TR-BE HSG/CC CDQ/8.

32. R. T. Snodgrass. *Developing Time-Oriented Database Applications in SQL*. Morgan Kaufmann, 1999.
33. S. Song and L. Chen. Discovering matching dependencies. In *CIKM*, 2009.
34. R. van der Meyden. The complexity of querying indefinite data about linearly ordered domains. *JCSS*, 54(1), 1997.
35. R. van der Meyden. Logical approaches to incomplete information: A survey. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*. Kluwer, 1998.
36. M. Weis and F. Naumann. Dogmatix tracks down duplicates in XML. In *SIGMOD*, 2005.
37. M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas. Guided data repair. *PVLDB*, 4(1), 2011.
38. H. Zhang, Y. Diao, and N. Immerman. Recognizing patterns in streams with imprecise timestamps. In *VLDB*, 2010.