Conditional Dependencies: A Principled Approach to Improving Data Quality

Wenfei Fan[1], Floris Geerts[2], and Xibei Jia[2,⋆]

[1] University of Edinburgh and Bell Laboratories
[2] University of Edinburgh

Abstract. Real-life data is often dirty and costs billions of pounds to businesses worldwide each year. This paper presents a promising approach to improving data quality. It effectively detects and fixes inconsistencies in real-life data based on conditional dependencies, an extension of database dependencies by enforcing bindings of semantically related data values. It accurately identifies records from unreliable data sources by leveraging relative candidate keys, an extension of keys for relations by supporting similarity and matching operators across relations. In contrast to traditional dependencies that were developed for improving the quality of schema, the revised constraints are proposed to improve the quality of data. These constraints yield practical techniques for data repairing and record matching in a uniform framework.

1 Introduction

Real-world data is often dirty: inconsistent, inaccurate, incomplete and/or stale. Dirty data may have disastrous consequences for everyone. Indeed, the following are real-life examples in the US, taken from [23]: (a) 800 houses in Montgomery County, Maryland, were put on auction block in 2005 due to mistakes in the tax payment data of Washington Mutual Mortgage. (b) Errors in a database of a bank led thousands of completed tax forms to be sent to wrong addresses in 2005, effectively helping identity thieves get hold of the names and bank account numbers of various people (c) The Internal Revenue Service (IRS) accused people for overdue tax caused by errors in the IRS database. There is no reason to believe that the scale of the problem is any different in the UK, or in any society that is dependent on information technology.

The costs and risks of dirty data are being increasingly recognized by all industries worldwide. Recent statistics reveal that enterprises typically find data error rates of approximately 1%–5%, and for some companies it is above 30% [28]. It is reported that dirty data costs US businesses 600 billion dollars annually [10], and that erroneously priced data in retail databases alone costs US consumers $2.5 billion each year [12]. It is also estimated that data cleaning accounts for 30%-80% of the development time and budget in most data warehouse projects [29].

⋆ Wenfei Fan and Floris Geerts are supported in part by EPSRC EP/E029213/1. Fan is also supported in part by a Yangtze River Scholar Award. Jia is supported by an RSE Fellowship.

A.P. Sexton (Ed.): BNCOD 2009, LNCS 5588, pp. 8–20, 2009.

While the prevalent use of the Web has made it possible to extract and integrate data from diverse sources, it has also increased the risks, on an unprecedented scale, of creating and propagating dirty data.

There has been increasing demand for data quality tools, to add accuracy and value to business processes. A variety of approaches have been put forward: probabilistic, empirical, rule-based, and logic-based methods. There have been a number of commercial tools for improving data quality, most notably ETL tools (extraction, transformation, loading), as well as research prototype systems, *e.g.,* Ajax, Potter's Wheel, Artkos and Telcordia (see [2,23] for a survey). Most data quality tools, however, are developed for a specific domain (*e.g.,* address data, customer records). Worse still, these tools often heavily rely on manual effort and low-level programs that are difficult to write and maintain [27].

To this end, integrity constraints yield a principled approach to improving data quality. Integrity constraints, *a.k.a.* data dependencies, are almost as old as relational databases themselves. Since Codd introduced functional dependencies in 1972, a variety of constraint formalisms have been proposed and widely used to improve *the quality of schema*, via normalization (see [13] for a survey). Recently, constraints have enjoyed a revival, for improving *the quality of data*.

Constraint-based methods specify data quality rules in terms of integrity constraints such that errors and inconsistencies in a database emerge as violations of the constraints. Compared to other approaches, constraint-based methods enjoy several salient advantages such as being declarative in nature and providing the ability to conduct inference and reasoning. Above all, constraints specify a fundamental part of the semantics of the data, and are capable of capturing semantic errors in the data. These methods have shown promise as a systematic method for reasoning about the semantics of the data, and for deducing and discovering data quality rules, among other things (see [7,14] for recent surveys).

This paper presents recent advances in constraint-based data cleaning. We focus on two problems central to data quality: (a) data repairing, to detect and fix inconsistencies in a database [1], and (b) record matching [11], to identify tuples that refer to the same real-world entities. These are undoubted top priority for every data quality tool. As an example [23], a company that operates drug stores successfully prevents at least one lawsuit per year that may result in at least a million dollars award, by investing on a tool that ensures the consistency between the medication histories of its customers and the data about prescription drugs. As another example, a recent effort to match records on licensed airplane pilots with records on individuals receiving disability benefits from the US Social Security Administration revealed forty pilots whose records turned up on both databases [23]. Constraints have proved useful in both data repairing and record matching.

For constraints to be effective for capturing errors and matching records in real-life data, however, it is necessary to revise or extend traditional database dependencies. Most work on constraint-based methods is based on traditional constraints such as functional dependencies and inclusion dependencies. These constraints were developed for schema design, rather than for data cleaning. They are not capable of detecting errors commonly found in real-life data.

In light of this, we introduce two extensions of traditional constraints, namely, conditional dependencies for capturing consistencies in real-life data (Section 2), and relative candidate keys for record matching (Section 3).

While constraint-based functionality is not yet available in commercial tools, practical methods have been developed for data cleaning, by using the revised constraints as data quality rules. We present a prototype data-quality system, based on conditional dependencies and relative candidate keys (Section 4).

The area of constraint-based data cleaning is a rich source of questions and vitality. We conclude the paper by addressing open research issues (Section 5).

2 Adding Conditions to Constraints

One of the central technical questions associated with data quality is how to characterize the consistency of data, *i.e.,* how to tell whether the data is clean or dirty. Traditional dependencies, such as functional dependencies (FDs) and inclusion dependencies (INDs), are required to hold on entire relation(s), and often fail to capture errors and inconsistencies commonly found in real-life data.

We circumvent these limitations by extending FDs and INDs through enforcing patterns of semantically related values; these patterns impose *conditions* on what part of the data the dependencies are to hold and which combinations of values should occur together. We refer to these extensions as *conditional functional dependencies* (CFDs) and *conditional inclusion dependencies* (CINDs), respectively.

2.1 Conditional Functional Dependencies

Consider the following relational schema for customer data:

customer (CC, AC, phn, name, street, city, zip)

where each tuple specifies a customer's phone number (country code CC, area code AC, phone phn), name and address (street, city, zip code). An instance D_0 of the customer schema is shown in Fig. 1.

Functional dependencies (FDs) on customer relations include:

$$f_1\text{: } [\mathsf{CC}, \mathsf{AC}, \mathsf{phn}] \rightarrow [\mathsf{street}, \mathsf{city}, \mathsf{zip}], \qquad f_2\text{: } [\mathsf{CC}, \mathsf{AC}] \rightarrow [\mathsf{city}].$$

That is, a customer's phone uniquely determines her address (f_1), and the country code and area code determine the city (f_2). The instance D_0 of Fig. 1 satisfies f_1 and f_2. In other words, if we use f_1 and f_2 to specify the consistency of customer data, *i.e.,* to characterize errors as violations of these dependencies, then no errors or inconsistencies are found in D_0, and D_0 is regarded clean.

A closer examination of D_0, however, reveals that none of its tuples is error-free. Indeed, the inconsistencies become obvious when the following constraints are considered, which intend to capture the semantics of customer data:

cfd_1: ([CC = 44, zip] → [street])
cfd_2: ([CC = 44, AC = 131, phn] → [street, city = 'EDI', zip])
cfd_3: ([CC = 01, AC = 908, phn] → [street, city = 'MH', zip])

	CC	AC	phn	name	street	city	zip
t_1:	44	131	1234567	Mike	Mayfield	NYC	EH4 8LE
t_2:	44	131	3456789	Rick	Crichton	NYC	EH4 8LE
t_3:	01	908	3456789	Joe	Mtn Ave	NYC	07974

Fig. 1. An instance of customer relation

Here cfd_1 asserts that for customers in the UK ($\mathsf{CC} = 44$), zip code uniquely determines street. In other words, cfd_1 is an "FD" that is to hold on the subset of tuples that satisfies the pattern "$\mathsf{CC} = 44$", *e.g.,* $\{t_1, t_2\}$ in D_0. It is not a traditional FD since it is defined with constants, and it is not required to hold on the entire customer relation D_0 (in the US, for example, zip code does not determine street). The last two constraints refine the FD f_1 given earlier: cfd_2 states that for any two UK customer tuples, if they have area code 131 and have the same phn, then they must have the same street and zip, and moreover, the city *must* be EDI; similarly for cfd_3.

Observe that tuples t_1 and t_2 in D_0 violate cfd_1: they refer to customers in the UK and have identical zip, but they differ in street. Further, while D_0 satisfies f_1, each of t_1 and t_2 in D_0 violates cfd_2: $\mathsf{CC} = 44$ and $\mathsf{AC} = 131$, but $\mathsf{city} \neq$ EDI. Similarly, t_3 violates cfd_3.

These constraints extend FDs by incorporating conditions, and are referred to as *conditional functional dependencies* (CFDs). CFDs are introduced in [16] to capture inconsistencies in a single relation. As shown by the example above, CFDs are capable of detecting errors and inconsistencies commonly found in real-life data sources that their traditional counterparts are not able to catch.

2.2 Conditional Inclusion Dependencies

We next incorporate conditions into inclusion dependencies (INDs). Consider the following two schemas, referred to as *source* and *target*:

Source: order (asin, title, type, price)

Target: book (isbn, title, price, format) CD (id, album, price, genre)

The source database contains a single relation order, specifying various types such as books, CDs, DVDs, ordered by customers. The target database has two relations, book and CD, specifying customer orders of books and CDs, respectively. Example source and target instances D_1 are shown in Fig. 2.

To detect errors across these databases, one might be tempted to use INDs:

$$\mathsf{order}\,(\mathsf{title}, \mathsf{price}) \subseteq \mathsf{book}\,(\mathsf{title}, \mathsf{price}), \qquad \mathsf{order}\,(\mathsf{title}, \mathsf{price}) \subseteq \mathsf{CD}\,(\mathsf{album}, \mathsf{price}).$$

One cannot expect, however, to find for each book item in the order table a corresponding CD item in the CD table; this might only hold *provided that* the book is an audio book. That is, there are certain inclusion dependencies from the source to the target, but only under certain *conditions*. The following *conditional inclusion dependencies* (CINDs) correctly reflect the situation:

cind_1: (order (title, price, type ='BOOK') $\subseteq$ book (title, price)),
cind_2: (order (title, price, type ='CD') $\subseteq$ CD (album, price)),
cind_3: (CD (album, price, genre ='a-book') $\subseteq$ book (title, price, format ='audio')).

Here cind_1 states that for each order tuple t, if its type is 'book', then there must exist a book tuple t' such that t and t' agree on their title and price attributes; similarly for cind_2. Constraint cind_3 asserts that for each CD tuple t, if its genre is 'a-book' (audio book), then there must be a book tuple t' such that the title and price of t' are identical to the album and price of t, and moreover, the format of t' must be 'audio'.

	asin	title	type	price
t_4:	a23	Snow White	CD	7.99
t_5:	a12	Harry Potter	book	17.99

(a) Example order data

	isbn	title	price	format
t_6:	b32	Harry Potter	17.99	hard-cover
t_7:	b65	Snow White	7.99	paper-cover

(b) Example book data

	id	album	price	genre
t_8:	c12	J. Denver	7.94	country
t_9:	c58	Snow White	7.99	a-book

(c) Example CD data

Fig. 2. Example order, book and CD data

While D_1 of Fig 2 satisfies cind_1 and cind_2, it violates cind_3. Indeed, tuple t_9 in the CD table has an 'a-book' genre, but it cannot find a match in the book table with 'audio' format. Note that the book tuple t_7 is not a match for t_9: although t_9 and t_7 agree on their album (title) and price attributes, the format of t_7 is 'paper-cover' rather than 'audio' as required by cind_3.

Along the same lines as CFDs, conditional inclusion dependencies (CINDs) are introduced [5], by extending INDs with conditions. Like CFDs, CINDs are required to hold only on a subset of tuples satisfying certain patterns. They are specified with constants, and cannot be expressed as standard INDs. CINDs are capable of capturing errors across different relations that traditional INDs cannot detect.

2.3 Extensions of Conditional Dependencies

CFDs and CINDs can be naturally extended by supporting disjunction and inequality. Consider, for example, customers in New York State, in which most cities (CT) have a *unique* area code, except NYC and LI (Long Island). Further, NYC area codes consist of 212, 718, 646, 347 and 917. One can express these as:

ecfd_1: $\mathsf{CT} \notin \{\text{NYC, LI}\} \rightarrow \mathsf{AC}$
ecfd_2: $\mathsf{CT} \in \{\text{NYC}\} \rightarrow \mathsf{AC} \in \{212, 718, 646, 347, 917\}$

where ecfd_1 asserts that the FD $\mathsf{CT} \rightarrow \mathsf{AC}$ holds if CT is *not in* the set {NYC, LI}; and ecfd_2 is defined with disjunction: it states that when CT is NYC, AC must be one of 212, 718, 646, 347, *or* 917.

An extension of CFDs by supporting disjunction and inequality has been defined in [4], referred to as eCFDs. This extension is strictly more expressive than CFDs. Better still, as will be seen shortly, the increased expressive power does not make our lives harder when it comes to reasoning about these dependencies.

2.4 Reasoning about Conditional Dependencies

To use CFDs and CINDs to detect and repair errors and inconsistencies, a number of fundamental questions associated with these conditional dependencies have to be settled. Below we address three most important technical problems, namely, the consistency, implication and axiomatizability of these constraints.

Consistency. Given a set Σ of CFDs (resp. CINDs), can one tell whether the constraints in Σ are dirty themselves? If the input set Σ is found inconsistent, then there is *no need* to check the data quality rules against the data at all. Further, the analysis helps the user discover errors in the rules.

This can be stated as the consistency problem for conditional dependencies. For a set Σ of CFDs (CINDs) and a database D, we write $D \models \Sigma$ if $D \models \varphi$ for *all* $\varphi \in \Sigma$. The *consistency problem* is to determine, given Σ defined on a relational schema $\mathcal{R}$, whether there exists a nonempty instance D of $\mathcal{R}$ such that $D \models \Sigma$.

One can specify arbitrary FDs and INDs without worrying about consistency. This is no longer the case for CFDs.

Example 1. Consider a set Σ_0 of CFDs of the form $([A = x \rightarrow [B] = \bar{x})$, where the domain of A is bool, x ranges over true and false, and $\bar{x}$ indicates the negation of x. Then there exists no nonempty instance D such that $D \models \Sigma_0$. Indeed, for any tuple t in D, no matter what $t[A]$ takes, CFDs in Σ_0 force $t[A]$ to take the other value from the finite domain bool. □

Table 1 compares the complexity bounds for the static analyses of CFDs, eCFDs and CINDs with their traditional counterparts. It turns out that while for CINDs the consistency problem is not an issue, for CFDs it is nontrivial. Worse, when CFDs and CINDs are put together, the problem becomes undecidable, as opposed to their trivial traditional counterpart. That is, the expressive power of CFDs and CINDs comes at a price of higher complexity for reasoning about them.

Implication. Another central technical problem is the *implication problem*: given a set Σ of CFDs (resp. CINDs) and a single CFD (resp. CIND) φ defined on a relational schema $\mathcal{R}$, it is to determine whether or not Σ entails φ, denoted by $\Sigma \models \varphi$, *i.e.,* whether for all instances D of $\mathcal{R}$, if $D \models \Sigma$ then $D \models \varphi$. Effective implication analysis allows us to deduce new cleaning rules and to remove redundancies from a given set of rules, among other things.

As shown in Table 1, the implication problem also becomes more intriguing for CFDs and CINDs than their counterparts for FDs and INDs.

In certain practical cases the consistency and implication analyses for CFDs and CINDs have complexity comparable to their traditional counterparts. As an example, for data cleaning in practice, the relational schema is often fixed, and only dependencies vary and are treated as the input. In this setting, the

Table 1. Complexity and finite axiomatizability

Dependencies	Consistency	Implication	Fin. Axiom
CFDs	NP-complete	coNP-complete	Yes
eCFDs	NP-complete	coNP-complete	Yes
FDs	$O(1)$	$O(n)$	Yes
CINDs	$O(1)$	EXPTIME-complete	Yes
INDs	$O(1)$	PSPACE-complete	Yes
CFDs + CINDs	undecidable	undecidable	No
FDs + INDs	$O(1)$	undecidable	No
in the absence of finite-domain attributes			
CFDs	$O(n^2)$	$O(n^2)$	Yes
CINDs	$O(1)$	PSPACE-complete	Yes
eCFDs	NP-complete	coNP-complete	Yes
CFDs + CINDs	undecidable	undecidable	No

consistency and implication problems for CFDs (resp. CINDs) have complexity similar to (resp. the same as) their traditional counterparts; similarly when no constraints are defined with attributes with a finite domain (*e.g.,* bool).

Axiomatizability. Armstrong's Axioms for FDs can be found in almost every database textbook, and are essential to the implication analysis of FDs. A finite set of inference rules for INDs is also in place. For conditional dependencies the finite axiomatizability is also important, as it reveals insight of the implication analysis and helps us understand how data quality rules interact with each other.

This motivates us to find a finite set $\mathcal{I}$ of inference rules that are *sound and complete* for implication analysis, *i.e.,* for any set Σ of CFDs (resp. CIND) and a single CFD (resp. CIND) φ, $\Sigma \models \varphi$ iff φ is provable from Σ using $\mathcal{I}$.

The good news is that when CFDs and CINDs are taken separately, they are finitely axiomatizable [16,5]. However, just like their traditional counterparts, when CFDs and CINDs are taken together, they are not finitely axiomatizable.

3 Extending Constraints with Similarity

Another central technical problem for data quality is record matching, *a.k.a.* record linkage, merge-purge, data deduplication and object identification. It is to identify tuples from (unreliable) relations that refer to the same real-world object. This is essential to data cleaning, data integration and credit-card fraud detection, among other things. Indeed, it is often necessary to correlate information about an object from multiple data sources, while the data sources may not be error free or may have different representations for the same object.

A key issue for record matching concerns how to determine matching keys [2,11], *i.e.,* what attributes should be selected and how they should be compared in order to identify tuples. While there has been a host of work on the topic, record matching tools often require substantial manual effort from human experts, or rely on probabilistic or learning heuristics (see [2,23,11] for surveys).

Constraints can help in automatically deriving matching keys from matching rules, and thus improve match quality and increase the degree of automation. To illustrate this, consider two data sources, specified by the following schemas:

$$\begin{array}{l}\mathsf{card}\ (\mathsf{c\#},\ \text{SSN},\ \text{FN},\ \text{LN},\ \mathsf{addr},\ \mathsf{tel},\ \mathsf{email},\ \mathsf{type}),\\ \mathsf{billing}\ (\mathsf{c\#},\ \text{FN},\ \text{SN},\ \mathsf{post},\ \mathsf{phn},\ \mathsf{email},\ \mathsf{item},\ \mathsf{price}).\end{array}$$

Here a card tuple specifies a credit card (number c# and type) issued to a card holder identified by SSN, FN (first name), LN (last name), addr (address), tel (phone) and email. A billing tuple indicates that the price of a purchased item is paid by a credit card of number c#, issued to a holder that is specified in terms of forename FN, surname SN, postal address post, phone phn and email.

Given an instance (D_c, D_b) of (card,billing), for *fraud detection*, one has to ensure that for any tuple $t \in D_c$ and $t' \in D_b$, if $t[\mathsf{c\#}] = t'[\mathsf{c\#}]$, then $t[Y_c]$ and $t'[Y_b]$ refer to the same holder, where $Y_c = [\text{FN}, \text{LN}, \mathsf{addr}, \mathsf{tel}, \mathsf{email}]$, and $Y_b = [\text{FN}, \text{SN}, \mathsf{post}, \mathsf{phn}, \mathsf{email}]$. Due to errors in the data sources, however, one may not be able to match $t[Y_c]$ and $t'[Y_b]$ via pairwise comparison of their attributes. Further, it is not feasible to manually select what attributes to compare. Indeed, to match tuples of arity n, there are 2^n possible comparison configurations.

One can leverage constraints and their reasoning techniques to derive "best" matching keys. Below are constraints expressing matching keys, which are an extension of relational keys and are referred to *relative candidate keys* (RCKs):

$$\begin{array}{ll}\mathsf{rck}_1\text{:} & \mathsf{card}[\text{LN}] = \mathsf{billing}[\text{SN}] \wedge \mathsf{card}[\mathsf{addr}] = \mathsf{billing}[\mathsf{post}] \wedge \mathsf{card}[\text{FN}] \approx \mathsf{billing}[\text{FN}]\\ & \quad \rightarrow \mathsf{card}[Y_c] \rightleftharpoons \mathsf{billing}[Y_b]\\ \mathsf{rck}_2\text{:} & \mathsf{card}[\mathsf{email}] = \mathsf{billing}[\mathsf{email}] \wedge \mathsf{card}[\mathsf{addr}] = \mathsf{billing}[\mathsf{post}] \rightarrow \mathsf{card}[Y_c] \rightleftharpoons \mathsf{billing}[Y_b]\\ \mathsf{rck}_3\text{:} & \mathsf{card}[\text{LN}] = \mathsf{billing}[\text{SN}] \wedge \mathsf{card}[\mathsf{tel}] = \mathsf{billing}[\mathsf{phn}] \wedge \mathsf{card}[\text{FN}] \approx \mathsf{billing}[\text{FN}]\\ & \quad \rightarrow \mathsf{card}[Y_c] \rightleftharpoons \mathsf{billing}[Y_b]\end{array}$$

Here rck_1 asserts that if $t[\text{LN}, \mathsf{addr}]$ and $t'[\text{SN}, \mathsf{post}]$ are identical and if $t[\text{FN}]$ and $t'[\text{FN}]$ are similar *w.r.t.* a similarity operator $\approx$, then $t[Y_c]$ and $t'[Y_b]$ match, *i.e.,* they refer to the same person; similarly for rck_2 and rck_3. Hence instead of comparing the entire Y_c and Y_b lists of t and t', one can inspect the attributes in rck_1–rck_3. If t and t' match on any of rck_1–rck_3, then $t[Y_c]$ and $t'[Y_b]$ *match*.

Better still, one can automatically derive RCKs from given matching rules. For example, suppose that rck_1 and the following matching rules are known, developed either by human experts or via learning from data samples: (a) if $t[\mathsf{tel}]$ and $t'[\mathsf{phn}]$ match, then $t[\mathsf{addr}]$ and $t'[\mathsf{post}]$ should refer to the same address (even if $t[\mathsf{addr}]$ and $t'[\mathsf{post}]$ might be radically different); and (b) if $t[\mathsf{email}]$ and $t'[\mathsf{email}]$ match, then $t[\text{FN}, \text{LN}]$ and $t'[\text{FN}, \text{SN}]$ match. Then rck_2 and rck_3 can be derived from these rules via automated reasoning.

The derived RCKs, when used as matching keys, can improve match quality: when t and t' differ in some pairs of attributes, *e.g.,* ([addr], [post]), they can still be matched via other, more reliable attributes, *e.g.,* ([LN,tel,FN], [SN,phn,FN]). In other words, true matches may be identified by derived RCKs, even when they cannot be found by the given matching rules from which the RCKs are derived.

In contrast to traditional constraints, RCKs are defined in terms of both equality and similarity; further, they are defined across multiple relations, rather than

on a single relation. Moreover, to cope with unreliable data, RCKs adopt a dynamic semantics very different from its traditional counterpart.

Several results have been established for RCKs [14]. (1) A finite inference system has been proposed for deriving new RCKs from matching rules. (2) A quadratic-time algorithm has been developed for deriving RCKs. (3) There are effective algorithms for matching records based on RCKs.

RCKs have also proved effective in improving the performance of record matching processes. It is often prohibitively expensive to compare every pair of tuples even for moderately sized relations [11]. To handle large data sources one often needs to adopt (a) blocking: partitioning the relations into blocks based on certain keys such that only tuples in the same block are compared, or (b) windowing: first sorting tuples using a key, and then comparing the tuples using a sliding window of a fixed size, such that only tuples within the same window are compared (see, *e.g.,* [11]). The match quality is highly dependent on *the choice of keys*. It has been experimentally verified that blocking and windowing can be effectively conducted by grouping similar tuples by (part of) RCKs.

4 Improving Data Quality with Dependencies

While constraints should logically become an essential part of data quality tools, we are not aware of any commercial tools with this facility. Nevertheless, we have developed a prototype system, referred to as SEMANDAQ, for improving the quality of relational data [17]. Based on conditional dependencies and relative candidate keys, the system has proved effective in repairing inconsistent data and matching non-error-free records when processing real-life data from two large US companies. Below we present some functionalities supported by SEMANDAQ.

Discovering data quality rules. To use dependencies as data quality rules, it is necessary to have techniques in place that can *automatically discover* dependencies from sample data. Indeed, it is often unrealistic to rely solely on human experts to design data quality rules via an expensive and long manual process.

This suggests that we settle *the profiling problem*. Given a database instance D, it is to find a *minimal cover* of all dependencies (*e.g.,* CFDs, CINDs) that hold on D, *i.e.,* a non-redundant set of dependencies that is logically equivalent to the set of all dependencies that hold on D. That is, we want to learn informative and interesting data quality rules from data, and prune away trivial and insignificant rules based on a threshold specified by users.

Several algorithms have been developed for discovering CFDs [6,18,22]. SEMANDAQ has implemented the discovery algorithms of [18].

Reasoning about data quality rules. A given set S of dependencies, either automatically discovered or manually designed by domain experts, may be dirty itself. In light of this we have to identify consistent dependencies from S, to be used as data quality rules. This problem is, however, nontrivial. As remarked in Section 2.4, it is already intractable to determine whether a given set S is consistent when S consists of CFDs only. It is also intractable to find a maximum subset of consistent rules from S.

Nevertheless, we have developed an approximation algorithm for finding a set S' of consistent rules from a set S of possibly inconsistent CFDs [16], while guaranteeing that S' is within a constant bound of the maximum consistent subset of S. SEMANDAQ supports this reasoning functionality.

Detecting errors. After a consistent set of data quality rules is identified, the next question concerns how to effectively catch errors in a database by using the rules. Given a set Σ of data quality rules and a database D, we want to *detect inconsistencies* in D, *i.e.,* to find all tuples in D that violate some rule in Σ.

We have shown that given a set Σ of CFDs and CINDs, a fixed number of SQL queries can be *automatically* generated such that, when being evaluated against a database D, the queries return all and only those tuples in D that violate Σ [4,16]. That is, we can effectively detect inconsistencies by leveraging existing facility of commercial relational database systems. This is another feature of SEMANDAQ.

Repairing errors. After the errors are detected, we want to automatically edit the data, fix the errors and make the data consistent. This is known as *data repairing* as formalized in [1]. Given a set Σ of dependencies and an instance D of a database schema $\mathcal{R}$, it is to find a candidate *repair* of D, *i.e.,* an instance D' of $\mathcal{R}$ such that D' satisfies Σ and D' *minimally differs* from the original database D [1]. This is the method that US national statistical agencies, among others, have been practicing for decades for cleaning census data [20,23].

Several repair models have been studied to assess the accuracy of repairs [1,3,8,30] (see [7] for a survey). It is shown, however, that the repairing problem is already intractable when traditional FDs or INDs are considered [3]. Nevertheless, repairing algorithms have been developed for FDs and INDs [3] and for CFDs [9].

SEMANDAQ supports these methods. It automatically generates candidate repairs and presents them to users for inspection, who may suggest changes to the repairs and the data quality rules. Based on users input, SEMANDAQ further improves the repairs until the users are satisfied with the quality of the repaired data. This interactive nature guarantees the *accuracy* of the repairs found.

Record matching. SEMANDAQ supports RCK-based methods outlined in Section 3, including (a) specification of matching rules, (b) automatic derivation of top k quality RCKs from a set of matching rules, for any given k, (c) record matching based on RCKs, and (d) blocking and windowing via RCKs. Compared to record matching facilities found in commercial tools, SEMANDAQ leverages RCKs to explore the semantics of the data and is able to find more accurate matches, while significantly reducing manual effort from domain experts.

5 Open Problems and Emerging Applications

The study of constraint-based data cleaning has raised as many questions as it has answered. While it yields a promising approach to improving data quality and will lead to practical data-quality tools, a number of open questions need to be settled. Below we address some of the open research issues.

The interaction between data repairing and record matching. Most commercial tools either support only one of these, or separate the two processes. However, these processes interact with each other and often need to be combined. The need is particularly evident in master data management (MDM), one of the fastest growing software markets [24]. In MDM for an enterprise, there is typically a collection of data that has already been cleaned, referred to as *master data* or *reference data*. To repair databases by capitalizing on available master data it is necessary to conduct record matching and data repairing at the same time.

To this end conditional dependencies and relative candidate keys allow us to conduct data repairing and record matching in a uniform constraint-based framework. This calls for the study of reasoning about conditional dependencies and relative candidate keys taken together, among other things.

Incomplete information. Incomplete information introduces serious problems to enterprises: it routinely leads to misleading analytical results and biased decisions, and accounts for loss of revenues, credibility and customers. Previous work either assumes a database to be closed (the Closed World Assumption, CWA, *i.e.,* all the tuples representing real-world entities are assumed already in place, but some data elements of the tuples may be missing), or open (the Open World Assumption, OWA, *i.e.,* a database may only be a proper subset of the set of tuples that represent real-world entities; see, *e.g.,* [25,26]). In practice, however, a database is often neither entirely closed nor entirely open. In MDM environment, for instance, master data is a closed database. Meanwhile a number of other databases may be in use, which may have missing tuples or missing data elements, but certain parts of the databases are *constrained by* the master data and are closed. To capture this we have proposed a notion of relative information completeness [15]. Nevertheless practical algorithms are yet to be developed to quantitatively assess the completeness of information *w.r.t.* user queries.

Repairing distributed data. In practice a relation is often fragmented, vertically or horizontally, and is distributed across different sites. In this setting, even inconsistency detection becomes nontrivial: it necessarily requires certain data to be shipped from one site to another. In other words, SQL-based techniques for detecting CFD violations no longer work. It is necessary to develop error detection and repairing methods for distributed data, to minimize data shipment. Another important yet challenging issue concerns the quality of data that is integrated from distributed, unreliable sources. While there has been work on automatically propagating data quality rules (CFDs) from data sources to integrated data [19], much more needs to be done to effectively detect errors during data integration and to propagate corrections from the integrated data to sources.

The quality of Web data. Data quality issues are on an even larger scale for data on the Web, *e.g.,* XML and semistructured data. Already hard for relational data, error detection and repairing are far more challenging for data on the Web. In the context of XML, for example, the constraints involved and their interaction with XML Schema are far more intriguing than their relational counterparts,

even for static analysis, let alone for data repairing. In this setting data quality remains, by and large, unexplored (see, *e.g.,* [21]). Another open issue concerns object identification, *i.e.,* to identify complex objects that refer to the same real-world entity, when the objects do not have a regular structure. This is critical not only to data quality, but also to Web page clustering, schema matching, pattern recognition, plagiarism detection and spam detection, among other things. Efficient techniques for identifying complex objects deserve a full exploration.

References

1. Arenas, M., Bertossi, L.E., Chomicki, J.: Consistent query answers in inconsistent databases. In: PODS (1999)
2. Batini, C., Scannapieco, M.: Data Quality: Concepts, Methodologies and Techniques. Springer, Heidelberg (2006)
3. Bohannon, P., Fan, W., Flaster, M., Rastogi, R.: A cost-based model and effective heuristic for repairing constraints by value modification. In: SIGMOD (2005)
4. Bravo, L., Fan, W., Geerts, F., Ma, S.: Increasing the expressivity of conditional functional dependencies without extra complexity. In: ICDE (2008)
5. Bravo, L., Fan, W., Ma, S.: Extending dependencies with conditions. In: VLDB (2007)
6. Chiang, F., Miller, R.: Discovering data quality rules. In: VLDB (2008)
7. Chomicki, J.: Consistent query answering: Five easy pieces. In: Schwentick, T., Suciu, D. (eds.) ICDT 2007. LNCS, vol. 4353, pp. 1–17. Springer, Heidelberg (2006)
8. Chomicki, J., Marcinkowski, J.: Minimal-change integrity maintenance using tuple deletions. Inf. Comput. 197(1-2), 90–121 (2005)
9. Cong, G., Fan, W., Geerts, F., Jia, X., Ma, S.: Improving data quality: Consistency and accuracy. In: VLDB (2007)
10. Eckerson, W.: Data quality and the bottom line: Achieving business success through a commitment to high quality data. The Data Warehousing Institute (2002)
11. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: A survey. TKDE 19(1), 1–16 (2007)
12. English, L.: Plain English on data quality: Information quality management: The next frontier. DM Review Magazine (April 2000)
13. Fagin, R., Vardi, M.Y.: The theory of data dependencies - An overview. In: ICALP (1984)
14. Fan, W.: Dependencies revisited for improving data quality. In: PODS (2008)
15. Fan, W., Geerts, F.: Relative information completeness. In: PODS (2009)
16. Fan, W., Geerts, F., Jia, X., Kementsietsidis, A.: Conditional functional dependencies for capturing data inconsistencies. TODS 33(2) (June 2008)
17. Fan, W., Geerts, F., Jia, X.: SEMANDAQ: A data quality system. based on conditional functional dependencies. In: VLDB, demo (2008)
18. Fan, W., Geerts, F., Lakshmanan, L., Xiong, M.: Discovering conditional functional dependencies. In: ICDE (2009)
19. Fan, W., Ma, S., Hu, Y., Liu, J., Wu, Y.: Propagating functional dependencies with conditions. In: VLDB (2008)
20. Fellegi, I., Holt, D.: A systematic approach to automatic edit and imputation. J. American Statistical Association 71(353), 17–35 (1976)

21. Flesca, S., Furfaro, F., Greco, S., Zumpano, E.: Querying and repairing inconsistent XML data. In: Ngu, A.H.H., Kitsuregawa, M., Neuhold, E.J., Chung, J.-Y., Sheng, Q.Z. (eds.) WISE 2005. LNCS, vol. 3806, pp. 175–188. Springer, Heidelberg (2005)
22. Golab, L., Karloff, H., Korn, F., Srivastava, D., Yu, B.: On generating near-optimal tableaux for conditional functional dependencies. In: VLDB (2008)
23. Herzog, T.N., Scheuren, F.J., Winkler, W.E.: Data Quality and Record Linkage Techniques. Springer, Heidelberg (2007)
24. Loshin, D.: Master Data Management, Knowledge Integrity Inc. (2009)
25. Imieliński, T., Lipski Jr., W.: Incomplete information in relational databases. J. ACM 31(4), 761–791 (1984)
26. van der Meyden, R.: Logical approaches to incomplete information: A survey. In: Chomicki, J., Saake, G. (eds.) Logics for Databases and Information Systems, pp. 307–356 (1998)
27. Rahm, E., Do, H.H.: Data cleaning: Problems and current approaches. IEEE Data Eng. Bull. 23(4), 3–13 (2000)
28. Redman, T.: The impact of poor data quality on the typical enterprise. Commun. ACM 41(2), 79–82 (1998)
29. Shilakes, C., Tylman, J.: Enterprise information portals. Merrill Lynch (1998)
30. Wijsen, J.: Database repairing using updates. TODS 30(3), 722–768 (2005)