A Uniform Dependency Language for Improving Data Quality

Wenfei Fan Floris Geerts
University of Edinburgh
{wenfei, fgeerts}inf.ed.ac.uk

Abstract

A variety of dependency formalisms have been studied for improving data quality. To treat these dependencies in a uniform framework, we propose a simple language, Quality Improving Dependencies (QIDs). We show that previous dependencies considered for data quality can be naturally expressed as QIDs, and that different enforcement mechanisms of QIDs yield various data repairing strategies.

1 Introduction

Data quality has been a longstanding line of research for decades, and has become one of the most pressing challenges for data management. As an example, it is estimated that dirty data costs US companies alone 600 billion dollars each year [11]. With this comes the need for studying techniques for improving data quality.

A variety of dependency formalisms have recently been studied to specify data quality rules, *e.g.,* functional dependencies (FDs) [1], conditional functional dependencies (CFDs) [6, 14], order dependencies (ODs) [22, 29, 23], currency dependencies (CDs) [17], sequential dependencies (SDs) [25], matching dependencies (MDs) [18, 13, 4] and editing rules (eRs) [19]. These dependencies have proved useful in detecting and correcting inconsistencies in relational data. However, given these different formalisms, one would naturally ask which dependencies should be used in an application? Is a data quality rule expressible in one formalism but not in another? Is it more expensive to use one formalism than another? How should dependencies be enforced to repair data? These suggest that we study these formalisms in a comparative basis.

This paper takes a first step toward answering these questions. We propose a simple dependency language, referred to as Quality Improving Dependencies (QIDs), and show that all the dependencies mentioned above can be expressed as QIDs. That is, QIDs provide a uniform language to characterize these formalisms. We also introduce different mechanisms for enforcing QIDs as data quality rules, via revisions of the chase process [1]. We show that data repairing algorithms [5, 9, 19, 20] are just implementations of these mechanisms.

In the rest of the paper, we introduce QIDs (Section 2), and show that dependencies studied for data quality are special cases of QIDs (Section 3). In addition, we present data quality problems in terms of QIDs (Section 4), emphasizing data repairing as QID enforcement. We also give an overview of recent work on these issues.

2 Quality Improving Dependencies

We define QIDs in terms of a notion of comparison relations, which compare pairs of tuples based on some of their characteristics. Below we first introduce comparison relations, and then define QIDs.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

2.1 Comparison Relations

Consider a database schema $\mathcal{R} = (R_1, \ldots, R_m)$, in which each relation schema R_i is specified with a set of attributes, denoted by $R_i\big(\mathsf{tid}, A_1, \ldots, A_{\mathsf{n}(i)}\big)$. Here tid is a *tuple identifier* and the domain of attribute A_j is denoted by $dom(A_j)$. We say that two attributes A_j and A_k are *compatible* if $dom(A_j) = dom(A_k)$. An instance $\mathcal{I}$ of $\mathcal{R}$ is $(I_1, \ldots, I_m)$, where for $i \in [1, m]$, I_i is finite set of tuples t with $t[A_j] \in dom(A_j)$ for each $A_j \in R_i$.

A comparison relation is simply a binary relation $\sim \subseteq dom(\mathsf{tid}) \times dom(\mathsf{tid})$. In practice, it is often used to group tuples (identified by their tid's) by comparing certain attributes (properties) of the tuples. More specifically, let X be a set of attributes $\{A_1, \ldots, A_k\}$, and $\mathcal{B}$ be a "binary" relation $\mathcal{B} \subseteq dom(X) \times dom(X)$ that compares the X attributes of tuples, where $dom(X) = dom(A_1) \times \cdots \times dom(A_k)$. A *comparison relation via* $(X, \mathcal{B})$, denoted by $\sim^X_{\mathcal{B}}$, is defined such that for any tuples t_1 and t_2 that contain attributes X, $\sim^X_{\mathcal{B}}\ (t_1[\mathsf{tid}], t_2[\mathsf{tid}])$ iff $\mathcal{B}(t_1[X], t_2[X])$. Let $\mathcal{I}$ be an instance of $\mathcal{R}$ and I be the set of all tuples in $\mathcal{I}$ that contain attributes X. Then

$$\sim^X_{\mathcal{B}} := \big\{(\mathsf{tid}_1, \mathsf{tid}_2) \mid t_i \in I,\ t_i[\mathsf{tid}] = \mathsf{tid}_i,\ i = 1, 2,\ \mathcal{B}(t_1[X], t_2[X])\big\}$$

for instance $\mathcal{I}$. That is, $\sim^X_{\mathcal{B}}$ relates pairs of tuples by only considering the information present in their X attributes. Observe that a pair of tuples related by $\sim^X_{\mathcal{B}}$ may possibly come from *different* relations in $\mathcal{I}$.

As an example, we consider comparison relations $\sim^A_{\mathcal{B}}$ defined on an attribute A, where $\mathcal{B}$ is a variation of the equality relation on A, *i.e.,* a binary relation in $dom(A) \times dom(A)$ that is reflexive, symmetric and transitive.

Example 1: One can define $\sim^A_{\mathcal{B}}$ to group tuples based on the equality of a certain attribute, via the following $\mathcal{B}$:
(1) $\mathsf{eq}_A = \{(a, a) \mid a \in dom(A)\}$, *i.e.,* for any pair (t_1, t_2) of tuples, $\sim^A_{\mathsf{eq}_A}\ (t_1[\mathsf{tid}], t_2[\mathsf{tid}])$ as long as t_1 and t_2 have the same A attribute value. For a specific constant $c \in dom(A)$, one can define $\mathcal{B}$ to be $\mathsf{eq}_c = \{(c, c)\}$, such that $\sim^A_{\mathsf{eq}_c}$ collects all tuples t in which attribute $t[A]$ is restricted to be the predefined constant c.
(2) $\mathsf{eq}_S = \cup_{c \in S} \mathsf{eq}_c$ for a (finite) set $S \subseteq dom(A)$, *i.e.,* $\sim^A_{\mathsf{eq}_S}$ groups those tuples t in which $t[A] \in S$. Similarly one may define $\mathcal{B}$ as $\mathsf{eq}_{\bar{S}} = (\mathsf{eq}_A \setminus \mathsf{eq}_S)$, such that $\sim^A_{\mathsf{eq}_{\bar{S}}}$ groups tuples t in which $t[A]$ does not belong to S.
(3) When the domain of A is ordered, we may define $\mathcal{B}$ as $\mathsf{eq}_{S(\mathsf{op}\, b)}$ where $S(\mathsf{op}\, b) = \{a \in dom(A) \mid a\, \mathsf{op}\, b\}$, and $\mathsf{op} \in \{<, \leqslant, >, \geqslant, =, \neq\}$; *i.e.,* $\sim^A_{\mathsf{eq}_{S(\mathsf{op}\, b)}}$ groups tuples t such that $t[A] \mathsf{op}\ b$. □

In practice, it is common to compare values of an attribute A based on their similarity rather than equality. In general, a similarity relation $\mathcal{B} \subseteq dom(A) \times dom(A)$ if it is reflexive and symmetric, but not necessarily transitive.

Example 2: Similarity relations are often derived from metrics. Suppose that A is an attribute whose domain is string. We define the similarity relation $\approx^{\mathsf{dl}}_{\theta}$ based on the Damerau-Levenstein metric (with threshold θ):

$$\approx^{\mathsf{dl}}_{\theta} = \{(s, t) \in \mathsf{string} \times \mathsf{string} \mid \text{edit distance}(s, t) \leqslant \theta\}.$$

In fact, any metric such as q-grams or Jaro distance can be turned into a similarity relation (see [10] for a survey on distance functions), from which a comparison relation (*e.g.,* $\sim^A_{\approx^{\mathsf{dl}}_{\theta}}$) can be readily derived. □

The comparison relations defined in terms of equality and similarity relations can be readily extended to a set X of attributes, *e.g.,* $\sim^X_{\mathsf{eq}_X}$ is defined as $\bigcap_{A \in X} \sim_{\mathsf{eq}_A}$, and $\sim^X_{\approx_X}$ as $\bigcap_{A \in X} \sim^A_{\approx_A}$.

A comparison relation $\sim^X_{\mathcal{B}}$ via $(X, \mathcal{B})$ is said to be *decomposable* if there exist $\mathcal{B}_i \subseteq dom(A_i) \times dom(A_i)$ such that $\sim^X_{\mathcal{B}} = \bigcap_{A_i \in X} \sim^{A_i}_{\mathcal{B}_i}$. For instance, the comparison relations $\sim^X_{\mathsf{eq}_X}$ and $\sim^X_{\approx_X}$ given above are clearly decomposable. Nevertheless, not all comparison relations are decomposable, as shown below.

Example 3: Let $X = \{A_1, \ldots, A_k\}$ be a set of ordered attributes, and let $\bar{a} = (a_1, \ldots, a_k)$ and $\bar{b} = (b_1, \ldots, b_k)$ be tuples in $dom(A_1) \times \cdots \times dom(A_k)$. Define $\bar{a}\, \mathsf{lexico}\, \bar{b}$ iff either $a_i = b_i$ for all $i \in [1, k]$ or there exist a $j \in [1, k]$ such that $a_j <_{A_j} b_j$ while $a_i = b_i$ for $i \in [1, j-1]$. Here $<_{A_j}$ denotes an order on A_j. Then the corresponding comparison relation $\sim^X_{\mathsf{lexico}}$ is not decomposable. Indeed, observe that $\sim^X_{\mathsf{lexico}}$ can be written as

$$\Big(\bigcap_{i \in [1,k]} \sim^{A_i}_{\mathsf{eq}}\Big) \cup \bigcup_{j \in [1,k]} \Big(\sim^{A_j}_{<_{A_j}} \cap \bigcap_{i \in [1, j-1]} \sim^{A_i}_{\mathsf{eq}}\Big);$$

in other words, it can be written as a union of decomposable comparison relations, but not as $\bigcap_{i \in [1,k]} \sim^{A_i}_{\mathcal{B}_i}$. This example also shows that comparison relations can be defined in terms of Boolean operations. □

2.2 Quality Improving Dependencies

We next define quality improving dependencies in terms of comparison relations. Let R and R' be two relations in $\mathcal{R}$. Let $X = \{A_1, \ldots, A_k\}$ and $X' = \{A'_1, \ldots, A'_k\}$ be attributes in R and R', respectively, such that A_i and A'_i are compatible for each $i \in [1, k]$; similarly for attributes $Y = \{E_1, \ldots, E_\ell\}$ and $Y' = \{E'_1, \ldots, E'_\ell\}$. Let $\sim^X_{\mathcal{B}_X}$ and $\sim^Y_{\mathcal{B}_Y}$ be comparison relations via $(X, \mathcal{B}_X)$ and $(Y, \mathcal{B}_Y)$, respectively.

A *quality improving dependency* (QID) defined on (R, R') is of the form

$$\varphi \;=\; \big(\sim^X_{\mathcal{B}_X}(R(X), R'(X')) \;\rightarrow\; \sim^Y_{\mathcal{B}_Y}(R(Y), R'(Y'))\big).$$

An instance $\mathcal{I}$ of $\mathcal{R}$ *satisfies* φ, denoted by $\mathcal{I} \models \varphi$, if

$$\forall\, \mathsf{tid} \in I, \mathsf{tid}' \in I' : \sim^X_{\mathcal{B}_X}(\mathsf{tid}, \mathsf{tid}') \;\longrightarrow \sim^Y_{\mathcal{B}_Y}(\mathsf{tid}, \mathsf{tid}'),$$

where I is the instance of R in $\mathcal{I}$, I' is the instance of R' in $\mathcal{I}$ for which attributes of X' and Y' are renamed as their counterparts in X and Y, respectively, *i.e.,* A'_i as A_i for $i \in [1, k]$ and B'_j as B_j for $j \in [1, \ell]$.

When $\sim^X_{\mathcal{B}_X}$ and $\sim^Y_{\mathcal{B}_Y}$ are decomposable as $(\mathcal{B}_1^{A_1}, \ldots, \mathcal{B}_k^{A_k})$ and $(\mathcal{B}_1^{E_1}, \ldots, \mathcal{B}_\ell^{E_\ell})$, respectively, then $\mathcal{I} \models \varphi$ if

$$\forall\, \mathsf{tid} \in I, \mathsf{tid}' \in I' : \bigwedge_{i \in [1,k]} \sim^{A_i}_{\mathcal{B}_i}(\mathsf{tid}, \mathsf{tid}') \;\longrightarrow \bigwedge_{j \in [1,\ell]} \sim^{E_i}_{\mathcal{B}_i}(\mathsf{tid}, \mathsf{tid}').$$

If all comparison relations in a QID φ are decomposable, we say that φ is *decomposable*.

When $R = R'$, $X = X'$ and $Y = Y'$, we say that φ is a *mono-QID*; otherwise we call it a *bi-QID*. In the sequel, we simply write a mono-QID as $R(\sim^X_{\mathcal{B}_X} \rightarrow \sim^Y_{\mathcal{B}_Y})$. We only consider birelational QIDs that relate tuples in one relation R to tuples in another (possibly the same) R'. In practice, two relations often suffice. Nonetheless, one can readily extend QIDs to multiple relations. When developing algorithms for discovering and validating QIDs, and for detecting and correcting errors based on QIDs, it is important to know whether the QIDs under consideration are decomposable or not, and whether one or two relations are required (see Section 4).

3 Capturing Existing Dependency Formalisms with QIDs

As examples of QIDs, we show that a variety of dependencies studied for data quality can be expressed as QIDs, as summarized in Table 3. We first consider formalisms that are expressible as *decomposable mono*-QIDs.

Functional dependencies (FDs) [1]. A functional dependency on a relation R is defined as $R(X \rightarrow Y)$, where X and Y are sets of attributes. An instance I of R satisfies the FD if for any two tuples t_1 and t_2 in I, $t_1[Y] = t_2[Y]$ whenever $t_1[X] = t_2[X]$, *i.e.,* the X attributes of a tuple uniquely determine its Y attributes.

Every FD $R(X \rightarrow Y)$ is a decomposable mono-QID $R(\sim^X_{\mathsf{eq}_X} \rightarrow \sim^Y_{\mathsf{eq}_Y})$, where $\sim^X_{\mathsf{eq}_X}$ is defined in Section 2.

Conditional functional dependencies (CFDs) [14]. CFDs extend FDs by incorporating constant patterns. More specifically, a CFD is defined as $R[(X \rightarrow Y) \mid t_p]$, where $R(X \rightarrow Y)$ is an FD, and t_p is a pattern tuple over $X \cup Y$. For each attribute $A \in X \cup Y$, $t_p[A] \in \mathit{dom}(A)$ or $t_p[A]$ is wildcard, denoted by $\sqcup$. A tuple t in an instance I of R is said to *match* t_p in X, denoted by $t[X] \asymp t_p[X]$, if for each $A \in X$, either $t_p[A] = \sqcup$ and $t[A]$ can be an arbitrary value in $\mathit{dom}(A)$, or $t[A] = t_p[A]$ when $t_p[A]$ is a constant. An instance I of R satisfies the CFD if for any two tuples t_1 and t_2 in I, if $t_1[X] = t_2[X] \asymp t_p[X]$ then also $t_1[Y] = t_2[Y] \asymp t_p[Y]$. For instance, $R[(\mathsf{CC}, \mathsf{zip} \rightarrow \mathsf{street}) \mid (44, \sqcup, \sqcup)]$ is a CFD, which states that when CC (country code) is 44 (indicating UK), zip code uniquely determines street.

Every CFD $R[(X \rightarrow Y) \mid t_p]$ is a decomposable mono-QID $R(\sim^X_{\mathsf{eq}_C} \rightarrow \sim^Y_{\mathsf{eq}_C})$, where $\sim^X_{\mathsf{eq}_C}$ is defined as $\bigcap_{A \in X} \sim^A_{\mathsf{eq}'_A}$. Here $\sim^A_{\mathsf{eq}'_A}$ is $\sim^A_{\mathsf{eq}_A}$ when $t_p[A] = \sqcup$, and $\sim^A_{\mathsf{eq}_c}$ if $t_p[A] = c$ (see Example 1); similarly for $\sim^Y_{\mathsf{eq}_C}$.

Extended CFDs (eCFDs) [6]. This class of dependencies extends CFDs $[R(X \rightarrow Y) \mid t_p]$ such that in the pattern tuples t_p, for each $A \in X \cup Y$, $t_p[A] = S$, $t_p[A] = \bar{S}$ for a finite set of constants $S \subseteq \mathit{dom}(A)$, or $t_p[A] = \sqcup$. We say that $t[X] \asymp t_p[X]$ if for each attribute $A \in X$, $t[A] \in S$ if $t_p[A] = S$, $t[A] \neq S$ if $t_p[A] = \bar{S}$, or $t_p[A] = \sqcup$. The semantics of eCFDs is the same as CFDs except for the revised matching operator.

Dependencies [References]	Quality Improving Dependencies		
	Mono/Bi	decomp.	QIDs (basic comparison relations used)
Functional dependencies (FD) [1]	Mono	Yes	$R(\sim^X_{\mathsf{eq}_X} \rightarrow \sim^Y_{\mathsf{eq}_Y})$ (defined with $\sim_{\mathsf{eq}_A}$)
Conditional FDs (CFD) [14]	Mono	Yes	$R(\sim^X_{\mathsf{eq}_C} \rightarrow \sim^Y_{\mathsf{eq}_C})$ (with $\sim_{\mathsf{eq}_A}$, $\sim_{\mathsf{eq}_a}$)
Extended CFDs (eCFD) [6]	Mono	Yes	$R(\sim^X_{\mathsf{eq}_S} \rightarrow \sim^Y_{\mathsf{eq}_S})$ ($\sim_{\mathsf{eq}_A}$, $\sim_{\mathsf{eq}_S}$,$\sim_{\mathsf{eq}_{\bar{S}}}$)
Similarity FD (SFD) [32, 34, 3, 30]	Mono	Yes	$R(\sim^X_{\approx_X} \rightarrow \sim^Y_{\approx_Y})$ ($\approx_X$, reflexive and symmetric)
Domain dependencies (DDs) [12]	Mono	Yes	$R(\emptyset \rightarrow \sim^A_{\asymp_S})$
Edits [21]	Mono	Yes	$R(\sim^X_{\asymp_S} \rightarrow \sim^{A_k}_{\asymp_{\bar{S}_k}})$ (with $\sim^A_{\asymp_S}$ and $\sim^A_{\asymp_{\bar{S}}}$)
Association rules (AR) [35, 28]	Mono	Yes	$R(\sim^X_{\mathsf{eq}_C} \rightarrow \sim^A_{\mathsf{eq}_a})$ (with $\sim_{\mathsf{eq}_a}$)
Order dependency (OD) [22, 29, 23]	Mono	Yes	$R(\sim^X_{\mathsf{op}_X} \rightarrow \sim^Y_{\mathsf{op}_Y})$ (op: partial orders)
Currency dependencies (CD) [17]	Mono	Yes	$R(\sim^X_{\mathcal{B}_X} \rightarrow \sim^A_{\prec_A})$ ($\sim^A_{\prec_A}$ a currency order)
Binary FDs (BFD) [33]	Bi	Yes	$\sim^X_{\mathsf{eq}_X} (R(X), R'(X')) \rightarrow \sim^Y_{\mathsf{eq}_Y} (R(Y), R'(Y'))$
Matching dependencies (MD) [18, 4]	Bi	Yes	$\sim^X_{\approx_X} (R(X), R'(X')) \rightarrow \sim^Y_{\mathsf{eq}_Y} (R(Y), R'(Y'))$
Editing rules (eR) [19]	Bi	Yes	$\sim^X_{\mathcal{B}_X} (R(X, X_p), R'(X', X_p)) \rightarrow \sim^B_{\mathsf{eq}_B} (R(B), R'(B'))$
Sequential dependencies (SD) [25]	Mono	No	$R(\sim^X_{\mathsf{succ}(<)} \rightarrow \sim^A_{\mathsf{gap}})$

Table 13: Different quality improving dependencies

Each eCFD $[R(X \rightarrow Y) \mid t_p]$ is also a decomposable mono-QID $R(\sim^X_{\mathsf{eq}_S} \rightarrow \sim^Y_{\mathsf{eq}_S})$. Here $\sim^X_{\mathsf{eq}_S}$ is defined as $\bigcap_{A \in X} \sim^A_{\mathsf{eq}'_A}$, where $\sim^A_{\mathsf{eq}'_A}$ is $\sim^A_{\mathsf{eq}_A}$ when $t_p[A] = \sqcup$, $\sim^A_{\mathsf{eq}_S}$ when $t_p[A] = S$, and it is $\sim^A_{\mathsf{eq}_{\bar{S}}}$ when $t_p[A] = \bar{S}$ (see Example 1 for the definitions of $\sim^A_{\mathsf{eq}_S}$ and $\sim^A_{\mathsf{eq}_{\bar{S}}}$); similarly for $\sim^Y_{\mathsf{eq}_S}$.

Similarity functional dependencies (SFDs) [32, 34, 3, 30]. Similarity FDs are a subclass of fuzzy FDs in which equality eq is relaxed to similarity relations that are reflexive and symmetric, but not necessarily transitive. More specifically, for each attribute A in R, a similarity relation $\approx_A$ on $dom(A)$ is defined. A similarity FD $R(X \rightarrow Y)$ is specified the same as an FD. It is interpreted along the same lines as FDs, except that it is based on similarity, and its satisfaction may be a "truth degree" in the range $[0, 1]$ rather than true or false.

Leveraging comparison relations $\sim^X_{\approx_X}$ ($\sim^Y_{\approx_Y}$) defined in Section 2, one can verify that each SFD $R(X \rightarrow Y)$ can be expressed as a decomposable mono-QID $R(\sim^X_{\approx_X} \rightarrow \sim^Y_{\approx_Y})$.

Domain dependencies (DDs)/Edits [12, 21]. A DD [12] is of the form $R(A, S)$, where A is an attribute and S is a set of constants taken from $dom(A)$. It aims to assure that for each tuple t in an instance of R, $t[A] \in S$. An edit e is of the form $R([A_1 : S_1] \times \cdots \times [A_k : S_k])$, where S_i is a set of constants in $dom(A_i)$ [21]. It states that for any tuple t in an instance of R, $t[A_1, \ldots, A_k] \notin S_1 \times \cdots \times S_k$. Observe that edits can be expressed as eCFDs in which pattern tuples contain no wildcards. In contrast not every such eCFD is an edit rule.

To express DDs and edits as QIDs, we use a relation $\asymp_S = \{(a, b) \in dom(A) \times dom(A) \mid a, b \in S\}$. Intuitively, $\asymp_S$ states that elements in S are indistinguishable. We also use $\asymp_{\bar{S}} = dom(A)^2 \setminus \asymp_S$, to identify elements not in S. We define comparison relation $\sim^A_{\asymp_S}$ and $\sim^A_{\asymp_{\bar{S}}}$ in terms of $\asymp_S$ and $\asymp_{\bar{S}}$, respectively.

A DD $R(A, S)$ can be expressed as a decomposable mono-QID $R(\emptyset \rightarrow \sim^A_{\asymp_S})$. An edit $R([A_1 : S_1] \times \cdots \times [A_k : S_k])$ can be specified as a decomposable mono-QID $R(\sim^X_{\asymp_S} \rightarrow \sim^{A_k}_{\asymp_{\bar{S}_k}})$, where $\sim^X_{\asymp_S} = \bigcap_{i \in [1,k-1]} \sim^{A_i}_{\asymp_{S_i}}$.

Association rules with 100% confidence (ARs) [35, 28]. An association rule (AR) is defined by means of an implication $R(([A_1 : a_1], \ldots, [A_k : a_k]) \rightarrow [B : b])$, where A_i and B are attributes, and a_i and b are constants from the corresponding domains. An instance I of R satisfies the AR with 100% confidence if for each tuple t in I, if $t[A_1, \ldots, A_k] = (a_1, \ldots, a_k)$ then $t[B] = b$. Such association rules are a special case of CFDs in which the pattern tuples consist of constants only. Hence, ARs are decomposable mono-QID defined in terms of $\sim^A_{\mathsf{eq}_a}$.

Order dependencies (ODs) [22, 29, 23]. Order dependencies (ODs) are specified as standard FDs in which equality may be replaced by a partial order relations $<, \leq, >, \geq$ on ordered attributes, as well as their complement relations (*e.g.,* $\leq^c$). More specifically, an OD is of the form $R(\tilde{X} \rightarrow \tilde{Y})$ in which $\tilde{X}$ and $\tilde{Y}$ are sets of attributes that are either unmarked (for $=$), or marked with one of the partial order relations. ODs can be expressed as decomposable mono-QIDs defined in terms of $\sim^A_{\mathsf{eq}_A}$ (if the attribute A is unmarked) and $\sim^A_{\mathsf{op}}$ (if A is marked).

Here for $\mathsf{op} \in \{<, \leq, >, \geq, \leq^c\}$, $\sim^A_{\mathsf{op}}$ is defined in terms of relation $\mathsf{op} = \{(a, b) \in \mathit{dom}(A) \times \mathit{dom}(A) \mid a \ \mathsf{op}\ b\}$.

Currency dependencies (CDs) [17]. Currency dependencies (CDs) are constraints of the form

$$\forall t_1, \ldots, t_k : R(\textstyle\bigwedge_{j\in[1,k]}(t_1[\mathsf{eid}] = t_j[\mathsf{eid}] \wedge \psi) \longrightarrow t_u \prec_A t_v),$$

where $u, v \in [1, k]$, $t_u \prec_A t_v$ indicates that $t_v[A]$ is more current than $t_u[A]$; and ψ is a conjunction of predicates of the form $t_j[A]\ \mathsf{op}\ t_h[A]$ with $\mathsf{op} \in \{=, \neq, <, \leq, >, \geq, \prec\}$, $t_j[A] = a$ or $t_j[A] \neq a$ for $a \in \mathit{dom}(A)$. Here eid denotes an entity id attribute, as introduced by Codd [8]. The CD is to assert that when t_i's are tuples pertaining to the same entity, and if condition ψ is satisfied, then tuple t_v must be more up-to-date than t_u in attribute A.

When $k = 2$, CDs can be expressed as decomposable mono-QIDs of the form $R(\sim^X_{\mathcal{B}_X} \rightarrow \sim^A_{\prec_A})$. Here $\sim^A_{\prec_A}$ is a comparison relation defined in terms of relation $\prec_A$, and $\sim^X_{\mathcal{B}_X}$ is an intersection of comparison relations $\sim^E_{\mathsf{op}}$ for $\mathsf{op} \in \{<, \leq, >, \geq, =, \neq\}$, and $\sim^E_{\mathsf{eq}_{S(\mathsf{op}\, a)}}$ for $\mathsf{op} \in \{=, \neq\}$, to express the condition ψ.

There are also *mono*-QIDs that are *not decomposable*.

Sequential dependencies (SDs) [25]. A sequential dependency (SD) is of the form $R(X \rightarrow_g A)$, where X is a set of ordered attributes, A is a numerical (linear ordered) attribute, and $g \subseteq \mathit{dom}(A)$ is an interval. It is to assert that for a predefined permutation π of tuples in an instance I of R that are increasing in X, *i.e.,* $t_{\pi(1)}[X] <_X t_{\pi(2)}[X] <_X \cdots <_X t_{\pi(N)}[X]$, we have that for any $i \in [1, N-1]$, $t_{\pi(i+1)}[A] - t_{\pi(i)}[A] \in g$. A conditional variant of sequential dependencies is studied [25] in which a sequential dependency is to hold only an a subset of tuples in I, where the subset is selected by means of admissible ranges of attribute values in X.

To express SDs as QIDs, we define the following two comparison relations: (1) $\sim^A_{\mathsf{gap}}$ in terms of relation $\mathit{gap} = \{(a, b) \in \mathit{dom}(A) \times \mathit{dom}(A) \mid b - a \in g\}$, which is neither reflexive, symmetric nor transitive; and (2) $\sim^X_{\mathsf{succ}(<)} = \sim^X_{\pi} \setminus (\sim^X_{\pi\_\mathsf{strict}} \cap (\sim^X_{\pi\_\mathsf{strict}})^{\circ})$, where $\sim^X_{\pi}$ is the comparison relation derived from the predefined order π, $\sim^X_{\pi\_\mathsf{strict}} = \sim^X_{\pi} \setminus \sim^X_{\mathsf{eq}}$, and S° denotes the reversal operation on binary relation S; observe that $\sim^X_{\mathsf{succ}(<)}$ is defined as a Boolean combination of comparison relations, and is not decomposable.

An SD $R(X \rightarrow_g A)$ can be readily expressed as a non-decomposable mono-QID $R(\sim^X_{\mathsf{succ}(<)} \rightarrow \sim^A_{\mathsf{gap}})$. In addition, for the conditional extension of SDs given in [25], one can construct an appropriate comparison relation $\sim^X_{\mathsf{range}}$ that restricts $\sim^X_{\mathsf{succ}(<)}$ and groups tuples together that fall in the admissible range.

We next consider *decomposable bi*-QIDs.

Binary FDs (BFDs) [27, 33]. A binary FD is an FD that relates attributes in two different relations. More specifically, a BFD is of the form $R[X] = R'[X'] \rightarrow R[Y] = R'[Y']$ where X, X' and Y, Y' are lists of pairwise compatible attributes in R and R', respectively. For an instance (I, I') of (R, R'), it is to assure that for any pair of tuples $t \in I$ and $t' \in I$, if $t[X] = t'[X']$ then also $t[Y] = t'[Y']$.

We can express BFDs as decomposable bi-QIDs of the form $\sim^X_{\mathsf{eq}_X}(R(X), R'(X')) \rightarrow \sim^Y_{\mathsf{eq}_Y}(R(Y), R'(Y'))$, where $\sim^X_{\mathsf{eq}_X}$ and $\sim^Y_{\mathsf{eq}_Y}$ are defined in terms of $\sim^A_{\mathsf{eq}}$ as given earlier for the case of FDs, but across R and R'.

Matching dependencies (MDs) [18, 13, 4]. Matching dependencies (MDs) were proposed to specify record matching rules in [18]. MDs were originally defined in terms of a dynamic semantics. Here we attempt to redefine MDs as QIDs, and interpret their dynamic semantics in terms of their enforcement strategies in Section 4.

When specified as QIDs, MDs can be viewed as an extension of both SFDs and BFDs. They compare certain attributes of tuples (possibly across different relations) in terms of similarity, and if these attributes are similar enough to each other, then identify some other attributes of the tuples. More specifically, an MD is of the form

$$(\bigwedge_{i\in[1,k]} R[A_i] \asymp_i R'[A'_i]) \longrightarrow (\bigwedge_{j\in[1,\ell]} R[B_j] = R'[B'_j]).$$

For an instance (I, I') of (R, R'), it assure that for any tuples $t \in I$ and $t' \in I'$, if $t[A_i] \asymp_i t'[A'_i]$ for $i \in [1, k]$, where $\asymp_i$ is a similarity relation, then $t[B_j]$ and $t'[B'_j]$ must be identified for $j \in [1, \ell]$, indicated by equality.

MDs can be written as decomposable bi-QIDs of the form $\sim^X_{\approx_X}(R(X), R'(X')) \rightarrow \sim^Y_{\mathsf{eq}_Y}(R(Y), R'(Y'))$, where $\sim^X_{\approx_X}$ and $\sim^Y_{\mathsf{eq}_Y}$ are given earlier for SFDs and FDs, respectively.

Editing rules (eRs) [19]. Editing rules (eRs) were introduced for correcting errors in a data relation (R) with accurate values from a master relation (R'). An eR is of the form $[(R(X), R'(X') \rightarrow R(B), R'(B')) \mid t_p[X_p]]$, where X and X' are lists of compatible attributes in schemas R and R', respectively, and $|X| = |X'|$. Moreover, B is an attribute of R but $B \notin X$, and B' is an attribute of R' but is not in X'. Finally, t_p is a pattern tuple over a set of distinct attributes X_p in R, where for each $A \in X_p$, $t_p[A]$ is either $\sqcup$ or a constant a drawn from $dom(A)$, similar to pattern tuples in CFDs. Intuitively, for an instance (I, I') of (R, R') and for any tuple $t \in I$, if $t[X_p] \asymp t_p[X_p]$, where $\asymp$ is the match relation defined for CFDs, and moreover, if there exists $t' \in I'$ such that $t[X] = t'[X']$, then we update $t[B]$ by letting $t[B] := t'[B]$, *i.e.,* taking the value $t'[B]$ from the master tuple t'.

Like MDs, eRs also have a dynamic semantics. Nevertheless, we can express eRs as QIDs in terms of a standard static semantics, by using equality to indicate updates. More specifically, an eR can be written as decomposable bi-QIDs of the form $\sim^X_{\mathcal{B}_X}\ (R(X, X_p), R'(X', X_p)) \rightarrow \sim^B_{\mathsf{eq}_B}\ (R(B), R'(B'))$, where $\sim^X_{\mathcal{B}_X}$ is defined as $\bigcap_{A \in X} \sim^A_{\mathsf{eq}_A} \cap \bigcap_{E \in X_p} \sim^E_{\mathsf{eq}'_E}$. Here $\sim^A_{\mathsf{eq}_A}$ is the equality relation on attributes of tuples across R and R', and $\sim^E_{\mathsf{eq}'_E}$ is defined along the same lines as its counterpart for CFDs on R attributes only, to enforce patterns.

4 Data Quality Problems

We have proposed QIDs as data quality rules, to catch and fix errors. Below we address several central data quality issues associated with QIDs, and provide an overview of recent advances in the study of these issues.

Discovering QIDs. To use QIDs as data quality rules, it is necessary to have techniques in place that can *automatically discover* QIDs from real-life data. Indeed, it is often unrealistic to rely solely on human experts to design data quality rules via an expensive manual process. This suggests that we settle the following problem.

Given a database $\mathcal{I}$, *the profiling problem* is to find a *minimal cover* of all QIDs that $\mathcal{I}$ satisfies, *i.e.,* a non-redundant set of QIDs that is logically equivalent to the set of all QIDs that hold on $\mathcal{I}$. We want to learn informative and interesting data quality rules from (possibly dirty) data, and prune away insignificant rules.

Several algorithms have been developed for discovering FDs (*e.g.,* [26]), CFDs [7, 15, 24] and MDs [31]. These algorithms allow a considerable pruning of the search space when QIDs are decomposable.

Validating QIDs. A given set Σ of QIDs, either automatically discovered or manually designed, may be inconsistent itself. In light of this we have to find those QIDs from Σ that are consistent and non-redundant, to be used as data quality rules. These highlight the need for studying the following classical problems for dependencies.

The *satisfiability problem* for QIDs is to determine, given a set Σ of QIDs defined on a relational schema $\mathcal{R}$, whether there exists a nonempty instance $\mathcal{I}$ of $\mathcal{R}$ such that $\mathcal{I} \models \Sigma$, *i.e.,* $\mathcal{I} \models \varphi$ for *each* QID $\varphi \in \Sigma$.

The *implication problem* for QIDs is to decide, given a set Σ of QIDs and a single QID φ defined on a relational schema $\mathcal{R}$, whether Σ entails φ, *i.e.,* whether for all instances $\mathcal{I}$ of $\mathcal{R}$, if $\mathcal{I} \models \Sigma$ then $\mathcal{I} \models \varphi$. Effective implication analysis allows us to remove redundancies and deduce new data quality rules from Σ.

These problems have been studied for FDs [1], CFDs [14], SFDs [3], edits [21], ODs [29], BFDs [27], MDs [13] and editing rules [19]. These issues are, however, nontrivial. When CFDs are concerned, for instance, both problems are intractable [14]. This calls for the study of effective heuristic algorithms for their analyses.

Error detection (localization). 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 QIDs and a database $\mathcal{I}$, we want to *detect inconsistencies* in $\mathcal{I}$, *i.e.,* to find all tuples and attributes in $\mathcal{I}$ that violate some QID in Σ. Error detection can be done more efficiently for decomposable mono-QIDs due to the absence of intra-relational joins.

Error detection methods have been developed for centralized databases [14] and distributed data [16], based on CFDs. Nevertheless, it is rather challenging to precisely localize errors in the data. Dependencies typically help us catch a pair of tuples (or attributes) that are inconsistent with each other, but fall short of telling us which attribute is erroneous. As an example, consider a simple CFD $R[(\mathsf{CC}, \mathsf{AC} \rightarrow \mathsf{city}) \mid (44, 131, \mathrm{Edi})]$, which states that in the UK (CC = 44), when the area code (AC) is 131, then the city must be Edinburgh. A tuple t: (CC = 44.

AC = 131, city = Ldn) violates this CFD: AC is 131 but the city is London. The CFD finds that $t[\mathsf{CC}]$, $t[\mathsf{AC}]$ or $t[\mathsf{city}]$ is incorrect, *i.e.,* an error is present in the tuple. However, it does not tell us which of the three attributes is wrong and to what value it should be changed. To rectify this problem, we need to adopt a stronger semantics when enforcing QIDs for data imputation, as will be seen below when we discuss editing rules [19].

Data repairing (data imputation). After the errors are detected, we want to automatically correct the errors and make the data consistent. This is known as *data repairing* [2]. Given a set Σ of QIDs and a database instance $\mathcal{I}$ of $\mathcal{R}$, it is to find a candidate *repair* of $\mathcal{I}$, *i.e.,* another instance $\mathcal{I}'$ of $\mathcal{R}$ such that $\mathcal{I}' \models \Sigma$ and $\mathsf{cost}(\mathcal{I}, \mathcal{I}')$ is minimum. Here $\mathsf{cost}()$ is a metric defined in terms of (a) the distance between $\mathcal{I}$ and $\mathcal{I}'$ and (b) the accuracy of attribute values in $\mathcal{I}$ (see, *e.g.,* [5, 9]). When a value v in $\mathcal{I}$ is changed to v' in $\mathcal{I}'$ in data repairing, the more accurate the original value v is and the more distant the new value v' is from v, the higher the cost of the change is. That is, we want to avoid changing accurate values in $\mathcal{I}$, and want the repair $\mathcal{I}'$ to *minimally differ* from $\mathcal{I}$.

No matter how important, the data repairing problem is nontrivial: it is intractable even when only a fixed set of FDs is considered [5]. In light of this, several heuristic algorithms have been developed, to repair data based on different strategies for enforcing FDs, CFDs, MDs and eRs [5, 9, 19, 20].

These QID enforcement strategies can be uniformly characterized in terms of a revised chase process (see, *e.g.,* [1] for details about chase). More specifically, assume that for each attribute $A \in \mathcal{R}$, a *fixing function* $\sigma_A : \mathit{dom}(\mathsf{tid}) \to \mathit{dom}(A)$ is defined. Given a tuple t in $\mathcal{I}$ identified by tid, the function is used to change the values of $t[A]$ to the value $\sigma_A(\mathsf{tid})$. Let $\bar{\sigma}$ denote a set of fixing functions σ_A, one for each $A \in \mathcal{R}$. Consider a QID $\varphi = (\sim^X_{\mathcal{B}_X} (R(X), R'(X')) \to \sim^Y_{\mathcal{B}_Y} (R(Y), R'(Y')))$, and two tuples t_1, t_2 in $\mathcal{I}$. We say that an instance $\mathcal{I}'$ of $\mathcal{R}$ is the *result of enforcing φ on t_1 and t_2 in $\mathcal{I}$ using $\bar{\sigma}$*, denoted by $\mathcal{I} \to^{\bar{\sigma}}_{\varphi,(t_1,t_2)} \mathcal{I}'$, if

- in $\mathcal{I}$: we have that $\sim^X_{\mathcal{B}_X} (t_1[\mathsf{tid}], t_2[\mathsf{tid}])$ holds, whereas $\sim^Y_{\mathcal{B}_Y} (t_1[\mathsf{tid}], t_2[\mathsf{tid}])$ does not hold;
- in $\mathcal{I}'$: $t_1[A_i] = \sigma_{A_i}(t_1[\mathsf{tid}])$, $t_2[A_i] = \sigma_{A_i}(t_2[\mathsf{tid}])$ for all $A_i \in X \cup Y$, and both $\sim^X_{\mathcal{B}_X} (t_1[\mathsf{tid}], t_2[\mathsf{tid}])$ and $\sim^Y_{\mathcal{B}_Y} (t_1[\mathsf{tid}], t_2[\mathsf{tid}]))$ hold; and finally,
- $\mathcal{I}$ and $\mathcal{I}'$ agree on every other tuple and attribute value.

For a set Σ of QIDs, we say that $\mathcal{I}'$ is a Σ-*repair* of $\mathcal{I}$ if there is a finite *chasing sequence* $\mathcal{I}_0 \ldots, \mathcal{I}_k$ such that

$$\mathcal{I}_0 = \mathcal{I} \longrightarrow^{\bar{\sigma}^1}_{\varphi^1,(t^1_1,t^1_2)} \mathcal{I}_1 \longrightarrow^{\bar{\sigma}^2}_{\varphi^2,(t^2_1,t^2_2)} \cdots \longrightarrow^{\bar{\sigma}^k}_{\varphi^k,(t^k_1,t^k_2)} \mathcal{I}_k = \mathcal{I}',$$

and furthermore, $\mathcal{I}' \models \Sigma$, where $\varphi^i \in \Sigma$ and $\bar{\sigma}^i$ is a set of fixing functions for $i \in [1, k]$. A Σ-repair $\mathcal{I}'$ of $\mathcal{I}$ is minimal if $\mathsf{cost}(\mathcal{I}, \mathcal{I}')$ is minimum. We say that an enforcement strategy of QIDs is *Church-Rosser* if for any set Σ of QIDs, all Σ-repairs obtained via different finite chasing sequences result in the same instance of $\mathcal{R}$.

A repairing algorithm is essentially an implementation of the "chase"-process, determined by (a) how fixing functions are defined, (b) in what order the QIDs in Σ are enforced, and (c) for each QID φ, how it is enforced on the tuples in $\mathcal{I}$ that violate φ. Below we outline how existing algorithms compute Σ-repairs by enforcing QIDs. Here decomposable QIDs again simplify the chase process because fixing functions can be defined independently for different attributes in a decomposable QID.

(Conditional) Functional dependencies [5, 9]. Given a set Σ of FDs and an instance $\mathcal{I}$, the repairing algorithm of [5] computes a Σ-repair by enforcing the FDs in Σ one by one, as follows. (1) For each FD $\varphi = R(\sim^X_{\mathsf{eq}_X} \to \sim^B_{\mathsf{eq}_B})$ in Σ and each tuple t in $\mathcal{I}$, it finds the set $V_{\varphi,t}$ of all tuples t' in $\mathcal{I}$ such that $\sim^X_{\mathsf{eq}_X} (t[\mathsf{tid}], t'[\mathsf{tid}])$ but not $\sim^Y_{\mathsf{eq}_Y} (t[\mathsf{tid}], t'[\mathsf{tid}])$. (2) It defines fixing functions $\bar{\sigma}$ such that for all $B \in Y$ and for all $t' \in V_{\varphi,t} \cup \{t\}$, $\sigma_B(t[\mathsf{tid}])$ is the same constant c, which is a value drawn from $\{t''[B] \mid t'' \in V_{\varphi,t} \cup \{t\}\}$ such that the total cost of changing $t'[B]$ to c is minimum. That is, it enforces φ on all tuples in $V_{\varphi,t}$ consecutively, by changing the values of their attributes in the right-hand side of φ to the same value. Furthermore, in order for the repairing process to terminate, a *hard constraint* is imposed, which requires that once σ_B identifies attributes of a pair of tuples, these attributes remain identified throughout the repairing process. Note that this still allows adjustments to the values of those attributes at a later stage, *i.e.,* different fixing functions $\bar{\sigma}^i$ may be used in the process. This enforcement strategy guarantees to find a Σ-repair for a set of FDs, but is not Church-Rosser.

This is extended by the algorithm of [9] for CFDs, in which fixing functions allow attribute values to be changed to a constant specified by a CFD, provided that the hard constraints imposed so far is not violated. For instance, tuple $t : (\mathsf{CC} = 44, \mathsf{AC} = 131, \mathsf{city} = \text{Ldn})$ can be changed to $(\mathsf{CC} = 44, \mathsf{AC} = 131, \mathsf{city} = \text{Edi})$ as specified by CFD $R[(\mathsf{CC}, \mathsf{AC} \to \mathsf{city}) \mid (44, 131, \text{Edi})]$, unless Ldn was assigned earlier by a fixing function σ_{city}. In the latter case, σ_{city} will not make any change to the right-hand side attribute, but instead a fixing function on the left-hand side attributes is applied; *e.g.,* t could be changed to $(\mathsf{CC} = 44, \mathsf{AC} = 020, \mathsf{city} = \text{Ldn})$ by σ_{AC}. Termination is also assured, but the enforcement strategy is not Church-Rosser.

Matching dependencies [4, 18]. MDs were proposed in [18] to infer matching rules. An enforcement strategy was introduced in [4] for repairing data with MDs. It enforces a set Σ of MDs one by one on an instance $\mathcal{I}$ as follows. Given an MD $\varphi = \sim^{X}_{\approx_X} (R(X), R'(X')) \to \sim^{Y}_{\mathsf{eq}_Y} (R(Y), R'(Y'))$, for each $B \in Y$, it employs matching functions $m_B : \mathit{dom}(B) \times \mathit{dom}(B) \to \mathit{dom}(B)$ that are idempotent ($m_B(b, b) = b$), commutative ($m_B(b, b') = m_B(b', b)$) and associative ($m_B(b, m_B(b', b'')) = m_B(m_B(b, b'), b'')$). For a pair (t_1, t_2) of tuples in $\mathcal{I}$ that violate φ, it defines fixing functions σ_B such that $\sigma_B(t_1[\mathsf{tid}]) = \sigma_B(t_2[\mathsf{tid}]) = m_B(t_1[B], t_2[B])$, and hence, computes $\mathcal{I} \to^{\bar{\sigma}}_{\varphi,(t_1,t_2)} \mathcal{I}'$, where $\bar{\sigma}$ consists of σ_B for all $B \in Y$. It is shown in [4] that m_B imposes a bounded lattice-structure on $\mathit{dom}(B)$ and that $\mathcal{I}'$ is an instance that dominates $\mathcal{I}$ in the corresponding bounded lattice on instances of $\mathcal{R}$. From this follows the termination of the process. The strategy is not Church-Rosser.

Employing both CFDs and MDs, a data repairing algorithm was developed in [20], with fixing functions defined for CFDs and MDs as above. It enforces a set of CFDs and MDs by interleaving data repairing and record matching operations; by leveraging their interaction, the algorithm improves the accuracy of repairs generated.

Editing rules [19]. As remarked earlier, dependencies typically only detect the presence of errors, but do not locate precisely what attributes are wrong and how to fix the errors. As a result, repairing algorithms often fail to correct errors and worse still, may even introduce new errors in the repairing process. To rectify this problem, a repairing strategy was proposed in [19] that aims to find a certain fix of a tuple, *i.e.,* a fix that is guaranteed correct, by using editing rules (eRs) and master data. Consider an eR $\varphi = \sim^{X}_{\mathcal{B}_X} (R(X, X_p), R'(X', X_p)) \to \sim^{B}_{\mathsf{eq}_B} (R(B), R'(B'))$, a tuple t_1 in a data instance I of R, and a tuple t_2 in a master relation of schema R' that provides consistent and correct data values. It defines a fixing function σ_B that equalizes $t_1[B]$ and $t_2[B]$ by always setting $\sigma_B(t_1[\mathsf{tid}])$ and $\sigma_B(t_2[\mathsf{tid}])$ to the master value $t_2[B]$. Furthermore, it allows $\mathcal{I} \to^{\sigma_B}_{\varphi,(t_1,t_2)} \mathcal{I}'$ by enforcing φ only if (a) $t_1[X, X_p]$ has been assured correct (*i.e.,* validated), either by users or via inference from fixes generated earlier; and (b) enforcing different eRs on t_1 and master tuples yields the same repair. In addition, after an attribute $t_1[B]$ is updated, it is no longer changed since it is already validated.

For instance, tuple $t : (\mathsf{CC} = 44, \mathsf{AC} = 131, \mathsf{city} = \text{Ldn})$ is changed to $(\mathsf{CC} = 44, \mathsf{AC} = 131, \mathsf{city} = \text{Edi})$ only if (a) $t[\mathsf{CC}, \mathsf{AC}, \mathsf{city}]$ is already validated, and (b) for all eRs φ in a given set Σ and all master tuples t' in I', either φ does not apply to t and t', or φ requires that $t[\mathsf{city}]$ and $t'[\mathsf{city}]$ be equalized and moreover, $t'[\mathsf{city}] = \text{Edi}$.

From these it follows that the enforcement strategy guarantees the termination of repairing processes. Furthermore, it is per definition Church-Rosser and assures that the fixes generated are certain.

These algorithms are just examples of the revised chase. We expect that different QID enforcement strategies could be developed along the same lines, and yield data repairing algorithms for a wider range of applications.

Acknowledgments. Fan is supported in part by the RSE-NSFC Joint Project Scheme, an IBM scalable data analytics for a smarter planet innovation award, and the National Basic Research Program of China (973 Program) 2012CB316200.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS*, 1999.

[3] R. Belohlávek and V. Vychodil. Data dependencies in Codd's relational model with similarities. In *Handbook of Research on Fuzzy Information Processing in Databases*. IGI Global, 2008.

[4] L. E. Bertossi, S. Kolahi, and L. V. S. Lakshmanan. Data cleaning and query answering with matching dependencies and matching functions. In *ICDT*, 2011.

[5] 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.

[6] L. Bravo, W. Fan, F. Geerts, and S. Ma. Increasing the expressivity of conditional functional dependencies without extra complexity. In *ICDE*, 2008.

[7] F. Chiang and R. Miller. Discovering data quality rules. In *VLDB*, 2008.

[8] E. F. Codd. Extending the database relational model to capture more meaning. *TODS*, 4(4):397–434, 1979.

[9] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving data quality: Consistency and accuracy. In *VLDB*, 2007.

[10] M. M. Deza and E. Deza. *Encyclopedia of Distances*. Springer, 2009.

[11] W. W. Eckerson. Data quality and the bottom line: Achieving business success through a commitment to high quality data. The Data Warehousing Institute, 2002.

[12] R. Fagin. A normal form for relational databases that is based on domains and keys. *TODS*, 6:387–415, 1981.

[13] W. Fan, H. Gao, X. Jia, J. Li, and S. Ma. Dynamic constraints for record matching. *The VLDB Journal*, 2011.

[14] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *TODS*, 33(2), 2008.

[15] W. Fan, F. Geerts, J. Li, and M. Xiong. Discovering conditional functional dependencies. *TKDE*, 23(5):683–698, 2011.

[16] W. Fan, F. Geerts, S. Ma, and H. Müller. Detecting inconsistencies in distributed data. In *ICDE*, 2010.

[17] W. Fan, F. Geerts, and J. Wijsen. Determining the currency of data. In *PODS*, 2011.

[18] W. Fan, X. Jia, J. Li, and S. Ma. Reasoning about record matching rules. *PVLDB*, 2(1):407–418, 2009.

[19] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Towards certain fixes with editing rules and master data. *PVLDB*, 3(1):173–184, 2010.

[20] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Interaction between record matching and data repairing. In *SIGMOD*, 2011.

[21] I. P. Fellegi and D. Holt. A systematic approach to automatic edit and imputation. *Journal of the American Statistical Association*, 71(353):17–35, 1976.

[22] S. Ginsburg and R. Hull. Order dependency in the relational model. *TCS*, 26(1-2):149 – 195, 1983.

[23] S. Ginsburg and R. Hull. Sort sets in the relational model. *J. ACM*, 33:465–488, 1986.

[24] L. Golab, H. Karloff, F. Korn, D. Srivastava, and B. Yu. On generating near-optimal tableaux for conditional functional dependencies. In *VLDB*, 2008.

[25] L. Golab, H. J. Karloff, F. Korn, A. Saha, and D. Srivastava. Sequential dependencies. *PVLDB*, 2(1):574–585, 2009.

[26] Y. Huhtala, J. Kärkk ainen, P. Porkka, and H. Toivonen. TANE: An efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, 42(2):100–111, 1999.

[27] E. Komissartschik. Restructuring and dependencies in databases. In *MFDBS*, 1989.

[28] M. Luxenburger. Implications partielles dans un contexte. *Mathématiques, Informatique et Sciences Humaines*, 29(113):35–55, 1991.

[29] W. Ng. Ordered functional dependencies in relational databases. *Inf. Syst.*, 24(7):535 – 554, 1999.

[30] K. V. S. V. N. Raju and A. K. Majumdar. Fuzzy functional dependencies and lossless join decomposition of fuzzy relational database systems. *TODS*, 13:129–166, 1988.

[31] S. Song and L. Chen. Discovering matching dependencies. In *CIKM*, 2009.

[32] M. I. Sözat and A. Yazici. A complete axiomatization for fuzzy functional and multivalued dependencies in fuzzy database relations. *Fuzzy Sets Syst.*, 117:161–181, 2001.

[33] J. Wang. Binary equality implication constraints, normal forms and data redundancy. *Inf. Process. Lett.*, 101(1):20–25, 2007.

[34] A. Yazici, E. Gocmen, B. Buckles, R. George, and F. Petry. An integrity constraint for a fuzzy relational database. In *FUZZ*, 1993.

[35] M. J. Zaki. Mining non-redundant association rules. *Data Min. Knowl. Discov.*, 9(3):223–248, 2004.