Data Quality: Theory and Practice

Data Quality: Theory and Practice

Wenfei Fan*

University of Edinburgh and Harbin Institute of Technology

Abstract. Real-life data are often dirty: inconsistent, inaccurate, incomplete, stale and duplicated. Dirty data have been a longstanding issue, and the prevalent use of Internet has been increasing the risks, in an unprecedented scale, of creating and propagating dirty data. Dirty data are reported to cost US industry billions of dollars each year. There is no reason to believe that the scale of the problem is any different in any other society that depends on information technology. With these comes the need for improving *data quality*, a topic as important as traditional data management tasks for coping with *the quantity* of the data.

We aim to provide an overview of recent advances in the area of data quality, from theory to practical techniques. We promote a conditional dependency theory for capturing data inconsistencies, a new form of dynamic constraints for data deduplication, a theory of relative information completeness for characterizing incomplete data, and a data currency model for answering queries with current values from possibly stale data in the absence of reliable timestamps. We also discuss techniques for automatically discovering data quality rules, detecting errors in real-life data, and for correcting errors with performance guarantees.

1 Data Quality: An Overview

Traditional database systems typically focus on *the quantity of data*, to support the creation, maintenance and use of large volumes of data. But such a database system may not find correct answers to our queries if the data in the database are "dirty", *i.e.,* when the data do not properly represent the real world entities to which they refer.

To illustrate this, let us consider an employee relation residing in a database of a company, specified by the following schema:

employee (FN, LN, CC, AC, phn, street, city, zip, salary, status)

Here each tuple specifies an employee's name (first name FN and last name LN), office phone (country code CC, area code AC, phone phn), office address (street, city, zip code), salary and marital status. An instance D_0 of the employee schema is shown in Figure 1.

* Fan is supported in part by EPSRC EP/J015377/1, the RSE-NSFC Joint Project Scheme, the 973 Program 2012CB316200 and NSFC 61133002 of China.

	FN	LN	CC	AC	phn	street	city	zip	salary	status
t_1:	Mike	Clark	44	131	null	Mayfield	NYC	EH4 8LE	60k	single
t_2:	Rick	Stark	44	131	3456789	Crichton	NYC	EH4 8LE	96k	married
t_3:	Joe	Brady	01	908	7966899	Mtn Ave	NYC	NJ 07974	90k	married
t_4:	Mary	Smith	01	908	7966899	Mtn Ave	MH	NJ 07974	50k	single
t_5:	Mary	Luth	01	908	7966899	Mtn Ave	MH	NJ 07974	50k	married
t_6:	Mary	Luth	44	131	3456789	Mayfield	EDI	EH4 8LE	80k	married

Fig. 1. An employee instance

Consider the following queries posted on relation D_0.

(1) Query Q_1 is to find the number of employees working in the NYC office (New York City). The answer to Q_1 in D_0 is 3, by counting tuples t_1, t_2 and t_3. However, the answer may not be correct, for the following reasons. First, the data in D_0 are *inconsistent*. Indeed, the CC and AC values of t_1, t_2 and t_3 have conflicts with their corresponding city attributes: when CC = 44 and AC = 131, the city should be Edinburgh (EDI) in the UK, rather than NYC; and similarly, when CC = 01 and AC = 908, city should be Murray Hill (MH) in the US. It is thus likely that NYC is not the true city value of t_1, t_2 and t_3. Second, the data in D_0 may be *incomplete* for employees working in NYC. That is, some tuples representing employees working in NYC may be *missing* from D_0. Hence we cannot trust 3 to be the answer to Q_1.

(2) Query Q_2 is to find the number of distinct employees with FN = Mary. In D_0 the answer to Q_2 is 3, by enumerating tuples t_4, t_5 and t_6. Nevertheless, the chances are that t_4, t_5 and t_6 actually refer to the same person: all these tuples were once the true values of Mary, but some have become obsolete. Hence the correct answer to Q_2 may be 1 instead of 3.

(3) Query Q_3 is to find Mary's current salary and current last name, provided that we know that t_4, t_5 and t_6 refer to the same person. Simply evaluating Q_3 on D_0 will get us that salary is either 50k or 80k, and that LN is either Smith or Luth. However, it does not tell us whether Mary's current salary is 50k, and whether her current last name is Smith. Indeed, reliable timestamps for t_4, t_5 and t_6 may not be available, as commonly found in practice, and hence, we cannot tell which of 50k or 80k is more current; similarly for LN.

This example tells us that when the data are dirty, we cannot expect a database system to answer our queries correctly, no matter what capacity it provides to accommodate large data and how efficient it processes our queries.

Unfortunately, real-life data are often *dirty*: inconsistent, duplicated, inaccurate, incomplete and/or out of date. Indeed, enterprises typically find data error rates of approximately 1%–5%, and for some companies it is above 30% [41]. In most data warehouse projects, data cleaning accounts for 30%-80% of the development time and budget [43], for improving the quality of the data rather than developing the systems. When it comes to incomplete information, it is

estimated that "pieces of information perceived as being needed for clinical decisions were missing from 13.6% to 81% of the time" [38]. When data currency is concerned, it is known that "2% of records in a customer file become obsolete in one month" [14]. 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.

Why do we care about dirty data? Data quality has become one of the most pressing challenges to data management. It is reported that dirty data cost US businesses 600 billion dollars annually [14], and that erroneously priced data in retail databases alone cost US consumers $2.5 billion each year [16]. While these indicate the daunting cost of dirty data in the US, there is no reason to believe that the scale of the problem is any different in any other society that is dependent on information technology. Dirty data have been a longstanding issue for decades, and the prevalent use of Internet has been increasing the risks, in an unprecedented scale, of creating and propagating dirty data.

These highlight the need for *data quality management*, to improve *the quality* of the data in our databases such that the data consistently, accurately, completely and uniquely represent the real-world entities to which they refer.

Data quality management is at least as important as traditional data management tasks for coping with *the quantity of data*. There has been increasing demand in industries for developing data-quality management systems, aiming to effectively detect and correct errors in the data, and thus to add accuracy and value to business processes. Indeed, the market for data-quality tools is growing at 16% annually, way above the 7% average forecast for other IT segments [34]. As an example, data quality tools deliver "an overall business value of more than 600 million GBP" each year at BT [40]. Data quality management is also a critical part of big data management, master data management (MDM) [37], customer relationship management (CRM), enterprise resource planning (ERP) and supply chain management (SCM), among other things.

This paper aims to highlight several central technical issues in connection with data quality, and to provide an overview of recent advances in data quality management. We present five important issues of data quality (Section 2), and outline a rule-based approach to cleaning dirty data (Section 3). Finally, we identify some open research problems associated with data quality (Section 4).

The presentation is informal, to incite curiosity in the study of data quality. We opt for breadth rather than depth in the presentation: important results and techniques are briefly mentioned, but the details are omitted. A survey of detailed data quality management techniques is beyond the scope of this paper, and a number of related papers are not referenced due to space constraints. We refer the interested reader to papers in which the results were presented for more detailed presentation of the results and techniques. In particular, we encourage the reader to consult [3,4,9,17,21] for recent surveys on data quality management. In fact a large part of this paper is taken from [21].

2 Central Issues of Data Quality

We highlight five central issues in connection with data quality: data consistency, data deduplication, data accuracy, information completeness and data currency.

2.1 Data Consistency

Data consistency refers to the validity and integrity of data representing real-world entities. It aims to detect inconsistencies or conflicts in the data. In a relational database, inconsistencies may exist within a single tuple, between different tuples in the same table, and between tuples across different relations.

As an example, consider tuples t_1, t_2 and t_3 in Figure 1. There are conflicts within each of these tuples, as well as inconsistencies between different tuples.

(1) It is known that in the UK (when $\mathsf{CC} = 44$), if the area code is 131, then the city should be Edinburgh (EDI). In tuple t_1, however, $\mathsf{CC} = 44$ and $\mathsf{AC} = 131$, but $\mathsf{city} \neq$ EDI. That is, there exist inconsistencies between the values of the CC, AC and city attributes of t_1; similarly for tuple t_2. These tell us that tuples t_1 and t_2 are erroneous.

(2) Similarly, in the US ($\mathsf{CC} = 01$), if the area code is 908, the city should be Murray Hill (MH). Nevertheless, $\mathsf{CC} = 01$ and $\mathsf{AC} = 908$ in tuple t_3, whereas its city is not MH. This indicates that tuple t_3 is not quite correct.

(3) It is also known that in the UK, zip code uniquely determines street. That is, for any two tuples that refer to employees in the UK, if they share the same zip code, then they should have the same value in their street attributes. However, while $t_1[\mathsf{CC}] = t_2[\mathsf{CC}] = 44$ and $t_1[\mathsf{zip}] = t_2[\mathsf{zip}]$, $t_1[\mathsf{street}] \neq t_2[\mathsf{street}]$. Hence there are conflicts between t_1 and t_2.

Inconsistencies in the data are typically identified as violations of *data dependencies* (*a.k.a.* integrity constraints [1]). Errors in a single relation can be detected by intrarelation constraints, while errors across different relations can be identified by interrelation constraints.

Unfortunately, traditional dependencies such as functional dependencies (FDs) and inclusion dependencies (INDs) fall short of catching inconsistencies commonly found in real-life data, such as the errors in tuples t_1, t_2 and t_3 above. This is not surprising: the traditional dependencies were developed for schema design, rather than for improving data quality.

To remedy the limitations of traditional dependencies in data quality management, conditional functional dependencies (CFDs [23]) and conditional inclusion dependencies (CINDs [7]) have recently been proposed, which extend FDs and INDs, respectively, by specifying patterns of semantically related data values. It has been shown that conditional dependencies are capable of capturing common data inconsistencies that FDs and INDs fail to detect. For example, the inconsistencies in t_1–t_3 given above can be detected by CFDs.

A theory of conditional dependencies is already in place, as an extension of classical dependency theory. More specifically, the satisfiability problem, implication problem, finite axiomatizability and dependency propagation have been studied for conditional dependencies, from the complexity to inference systems to algorithms. We refer the interested reader to [7,6,23,31] for details.

2.2 Data Deduplication

Data deduplication aims to identify tuples in one or more relations that refer to the same real-world entity. It is also known as entity resolution, duplicate detection, record matching, record linkage, merge-purge, database hardening, and object identification (for data with complex structures).

For example, consider tuples t_4, t_5 and t_6 in Figure 1. To answer query Q_2 given earlier, we want to know whether these tuples refer to the same employee. The answer is affirmative if, for instance, there exists another relation which indicates that Mary Smith and Mary Luth have the same email account.

The need for studying data deduplication is evident: for data cleaning it is needed to eliminate duplicate records; for data integration it is to collate and fuse information about the same entity from multiple data sources; and for master data management it helps us identify links between input tuples and master data. The need is also highlighted by payment card fraud, which cost $4.84 billion worldwide in 2006 [42]. In fraud detection it is a routine process to cross-check whether a credit card user is the legitimate card holder. As another example, there was a recent effort to match records on licensed airplane pilots with records on individuals receiving disability benefits from the US Social Security Administration. The finding was quite surprising: there were forty pilots whose records turned up in both databases (cf. [36]).

No matter how important it is, data deduplication is nontrivial. Indeed, tuples pertaining to the same object may have different representations in various data sources with different schemas. Moreover, the data sources may contain errors. These make it hard, if not impossible, to match a pair of tuples by simply checking whether their attributes pairwise equal. Worse still, it is often too costly to compare and examine every pair of tuples from large data sources.

Data deduplication is perhaps the most extensively studied data quality problem. A variety of approaches have been proposed: probabilistic, learning-based, distance-based, and rule-based (see [15,36,39] for recent surveys).

We promote a dependency-based approach for detecting duplicates, which allows us to capture the interaction between data deduplication and other aspects of data quality in a uniform logical framework. To this end a new form of dependencies, referred to as *matching dependencies*, has been proposed for data deduplication [18]. These dependencies help us decide *what attributes to compare* and *how to compare these attributes* when matching tuples. They allow us to deduce alternative attributes to inspect such that when matching cannot be done by comparing attributes that contain errors, we may still find matches by using other, more reliable attributes.

In contrast to traditional dependencies that we are familiar with such as FDs and INDs, matching dependencies are dynamic constraints: they tell us what data have to be updated as a consequence of record matching. A dynamic constraint theory has been developed for matching dependencies, from deduction analysis to finite axiomatizability to inference algorithms (see [18] for details).

2.3 Data Accuracy

Data accuracy refers to the closeness of values in a database to the true values of the entities that the data in the database represent. Consider, for example, a person schema:

person (FN, LN, age, height, status)

where each tuple specifies the name (FN, LN), age, height and marital status of a person. An instance of person is shown below, in which s_0 presents the "true" information for Mike.

	FN	LN	age	height	status
s_0:	Mike	Clark	14	1.70	single
s_1:	M.	Clark	14	1.69	married
s_2:	Mike	Clark	45	1.60	single

Given these, we can conclude that the values of $s_1[\mathsf{age}, \mathsf{height}]$ are more accurate than $s_2[\mathsf{age}, \mathsf{height}]$, as they are closer to the true values for Mike, while $s_2[\mathsf{FN}, \mathsf{status}]$ are more accurate than $s_1[\mathsf{FN}, \mathsf{status}]$. It is more challenging, however, to determine the relative accuracy of s_1 and s_2 when the reference s_0 is unknown, as commonly found in practice. In this setting, it is still possible to find that for certain attributes, the values in one tuple are more accurate than another by an analysis of the semantics of the data, as follows.

(1) Suppose that we know that Mike is still going to middle school. From this, we can conclude that $s_1[\mathsf{age}]$ is more accurate than $s_2[\mathsf{age}]$. That is, $s_1[\mathsf{age}]$ is closer to Mike's true age value than $s_2[\mathsf{age}]$, although Mike's true age may not be known. Indeed, it is unlikely that students in a middle school are 45 years old. Moreover, from the age value ($s_1[\mathsf{age}]$), we may deduce that $s_2[\mathsf{status}]$ may be more accurate than $s_1[\mathsf{status}]$.

(2) If we know that $s_1[\mathsf{height}]$ and $s_2[\mathsf{height}]$ were once correct, then we may conclude that $s_1[\mathsf{height}]$ is more accurate than $s_2[\mathsf{height}]$, since the height of a person is typically monotonically increasing, at least when the person is young.

2.4 Information Completeness

Information completeness concerns whether our database has complete information to answer our queries. Given a database D and a query Q, we want to know whether Q can be completely answered by using only the data in D. If the information in D is incomplete, one can hardly expect its answer to Q to be accurate or even correct.

In practice our databases often do not have sufficient information for our tasks at hand. For instance, the value of $t_1[\mathsf{phn}]$ in relation D_0 of Figure 1 is missing, as indicated by null. Worse still, tuples representing employees may also be missing from D_0. As we have seen earlier, for query Q_1 given above, if some tuples representing employees in the NYC office are missing from D_0, then the answer to Q_1 in D_0 may not be correct. 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.

How should we cope with incomplete information? Traditional work on information completeness adopts either the Closed World Assumption (CWA) or the Open World Assumption (OWA), stated as follows (see, *e.g.,* [1]).

- The CWA assumes that a database has collected all the tuples representing real-world entities, but some *attribute values* of the tuples may be *missing*.
- The OWA assumes that in addition to missing values, some *tuples* representing real-world entities may also be *missing*. That is, our database may only be a proper subset of the set of tuples that represent real-world entities.

Database textbooks typically tell us that the world is closed: all the real-world entities of our interest are assumed already represented by tuples residing in our database. After all, database theory is typically developed under the CWA, which is the basis of negation in our queries: a fact is viewed as false unless it can be proved from explicitly stated facts in our database.

Unfortunately, in practice one often finds that not only attribute values but also tuples are missing from our database. That is, the CWA is often too strong to hold in the real world. On the other hand, the OWA is too weak: under the OWA, we can expect few sensible queries to find complete answers.

The situation is not as bad as it seems. In the real world, neither the CWA nor the OWA is quite appropriate in emerging applications such as master data management. In other words, real-life databases are *neither* entirely closed-world *nor* entirely open-world. Indeed, an enterprise nowadays typically maintains *master data* (*a.k.a. reference data*), a single repository of high-quality data that provides various applications with a synchronized, consistent view of the core business entities of the enterprise (see, *e.g.,* [37], for master data management). The master data contain complete information about the enterprise in certain categories, *e.g.,* employees, departments, projects, and equipment. Master data can be regarded as a closed-world database for the core business entities of the enterprise. Meanwhile a number of other databases may be in use in the enterprise for, *e.g.,* sales, project control and customer support. On one hand, the information in these databases may not be complete, *e.g.,* some sale transaction records may be missing. On the other hand, certain parts of the databases are *constrained by* the master data, *e.g.,* employees and projects. In other words, these databases are *partially closed*. The good news is that we often find that partially closed databases have complete information to answer our queries at hand.

To rectify the limitations of the CWA and the OWA, a theory of relative information completeness has been proposed [20,19], to specify partially closed

databases *w.r.t.* available master data. In addition, several fundamental problems in connection with relative completeness have been studied, to determine whether our database has complete information to answer our query, and when the database is incomplete for our tasks at hand, to decide what additional data should be included in our database to meet our requests. The complexity bounds of these problems have been established for various query languages.

2.5 Data Currency

Data currency is also known as *timeliness*. It aims to identify the current values of entities represented by tuples in a database that may contain stale data, and to answer queries with the current values.

The question of data currency would be trivial if all data values carried valid timestamps. In practice, however, one often finds that timestamps are unavailable or imprecise [46]. Add to this the complication that data values are often copied or imported from other sources [12,13], which may not support a uniform scheme of timestamps. These make it challenging to identify the "latest" values of entities from the data in our database.

For example, recall query Q_3 and the employee relation D_0 of Figure 1 given earlier. Assume that tuples t_4, t_5 and t_6 are found pertaining to the same employee Mary by data deduplication. As remarked earlier, in the absence of reliable timestamps, the answer to Q_3 in D_0 does not tell us whether Mary's current salary is 50k or 80k, and whether her current last name is Smith or Luth.

Not all is lost. In practice it is often possible to deduce currency orders from the semantics of the data, as illustrated below.

(1) While we do not have timestamps associated with Mary's salary, we know that the salary of each employee in the company does *not* decrease, as commonly found in the real world. This tells us that t_6[salary] is more current than t_4[salary] and t_5[salary]. Hence we may conclude that Mary's current salary is 80k.

(2) We know that the marital status can only change from single to married and from married to divorced; but not from married to single. In addition, employee tuples with the most current marital status also contain the most current last name. Therefore, t_6[LN] = t_5[LN] is more current than t_4[LN]. From these we can infer that Mary's current last name is Luth.

A data currency model has recently been proposed in [26], which allows us to specify and deduce data currency when temporal information is only partly known or not available at all. Moreover, a notion of certain current query answers is introduced there, to answer queries with current values of entities derived from a possibly stale database. In this model the complexity bounds of fundamental problems associated with data currency have been established, for identifying the current value of an entity in a database in the absence of reliable timestamps, answering queries with current values, and for deciding what data should be imported from other sources in order to answer query with current values. We encourage the interested reader to consult [26] for more detailed presentation.

2.6 Interactions between Data Quality Issues

To improve data quality we often need to deal with each and every of the five central issues given above. Moreover, there issues interact with each other, as illustrated below.

As we have seen earlier, tuples t_1, t_2 and t_3 in the relation D_0 of Figure 1 are inconsistent. We show how data deduplication may help us resolve the inconsistencies. Suppose that the company maintains a master relation for its offices, consisting of consistent, complete and current information about the address and phone number of each office. The master relation is specified by schema:

$$\mathsf{office}\ (\mathsf{CC},\ \mathsf{AC},\ \mathsf{phn},\ \mathsf{street},\ \mathsf{city},\ \mathsf{zip}),$$

and is denoted by D_m, given as follows:

	CC	AC	phn	street	city	zip
t_{m1}:	44	131	3456789	Mayfield	EDI	EH4 8LE
t_{m2}:	01	908	7966899	Mtn Ave	MH	NJ 07974

Then we may "clean" t_1, t_2 and t_3 by leveraging the interaction between data deduplication and data repairing processes (for data consistency) as follows.

(1) If the values of the CC, AC attributes of these tuples are confirmed accurate, we can safely update their city attributes by letting $t_1[\mathsf{city}] = t_2[\mathsf{city}] := \text{EDI}$, and $t_3[\mathsf{city}] := \text{MH}$, for reasons remarked earlier. This yields t'_1, t'_2 and t'_3, which differ from t_1, t_2 and t_3, respectively, only in their city attribute values.

(2) We know that if an employee tuple $t \in D_0$ and an office tuple $t_m \in D_m$ agree on their address $(\mathsf{street}, \mathsf{city}, \mathsf{zip})$, then the two tuples "match", *i.e.,* they refer to the same address. Hence, we can update $t[\mathsf{CC}, \mathsf{AC}, \mathsf{phn}]$ by taking the corresponding master values from t_m. This allows us to change $t'_2[\mathsf{street}]$ to $t_{m1}[\mathsf{street}]$. That is, we repair $t'_2[\mathsf{street}]$ by matching t'_2 and t_{m1}. This leads to tuple t''_2, which differs from t'_2 only in the street attribute.

(3) We also know that for employee tuples t_1 and t_2, if they have the same address, then they should have the same phn value. In light of this, we can augment $t'_1[\mathsf{phn}]$ by letting $t'_1[\mathsf{phn}] := t''_2[\mathsf{phn}]$, and obtain a new tuple t''_1.

One can readily verify that t''_1, t''_2 and t'_3 are consistent. In the process above, we "interleave" operations for resolving conflicts (steps 1 and 3) and operations for detecting duplicates (step 2). On one hand, conflict resolution helps deduplication: step 2 can be conducted only after $t_2[\mathsf{city}]$ is corrected. On the other hand, deduplication also helps us resolve conflicts: $t'_1[\mathsf{phn}]$ is enriched only after $t'_2[\mathsf{street}]$ is fixed via matching.

There are various interactions between data quality issues, including but not limited to the following.

- Data currency can be improved if more temporal information can be obtained in the process for improving information completeness.

- To determine the current values of an entity, we need to identify tuples pertaining to the same entity, via data deduplication. For instance, to find Mary's LN in relation D_0 of Figure 1, we have to ask whether tuples t_4, t_5 and t_6 refer to the same person.
- To resolve conflicts in tuples representing an entity, we have to determine whether the information about the entity is complete, and only if so, we can find the true value of the entity from the data available in our database.

These suggest that a practical data quality management system should provide functionality to deal with each and every of five central issues given above, and moreover, leverage the interactions between these issues to improve data quality. There has been preliminary work on the interaction between data deduplication and data repairing [27], as illustrated by the example above.

3 Improving Data Quality

Real-life data are often dirty, and dirty data are costly. In light of these, effective techniques have to be in place to improve the quality of our data. But how?

Errors in Real-Life Data. To answer this question, we first classify errors typically found in the real world. There are two types of errors, namely, syntactic errors and semantic errors, as illustrated below.

(1) Syntactic errors: violations of domain constraints by the values in our database. For example, name = 1.23 is a syntactic error if the domain of attribute name is string, whereas the value is numeric. Another example is age = 250 when the range of attribute age is [0, 120].

(2) Semantic errors: discrepancies between the values in our database and the true values of the entities that our data intend to represent. All the examples we have seen in the previous sections are semantic errors, related to data consistency, deduplication, accuracy, currency and information completeness.

While syntactic errors are relatively easy to catch, it is far more challenging to detect and correct semantic errors. Below we focus on semantic errors.

Dependencies as Data Quality Rules. A central question concerns how we can tell whether our data have semantic errors, *i.e.,* whether the data are dirty or clean? To this end, we need data quality rules to detect semantic errors in our data and fix those errors. But what data quality rules should we adopt?

A natural idea is to use data dependencies (*a.k.a.* integrity constraints). Dependency theory is almost as old as relational databases themselves. Since Codd [10] introduced functional dependencies, a variety of dependency languages, defined as various classes of first-order (FO) logic sentences, have been developed. There are good reasons to believe that dependencies should play an important role in data quality management systems. Indeed, dependencies specify a fundamental part of the semantics of data, in a declarative way, such that errors emerge as violations of the dependencies. Furthermore, inference systems,

implication analysis and profiling methods for dependencies have shown promise as a systematic method for reasoning about the semantics of the data. These help us deduce and discover rules for improving data quality, among other things. In addition, all the five central aspects of data quality – data consistency, deduplication, accuracy, currency and information completeness – can be specified in terms of data dependencies. This allows us to treat various data quality issues in a uniform logical framework, in which we can study their interactions.

Nevertheless, to make practical use of dependencies in data quality management, classical dependency theory has to be extended. Traditional dependencies were developed to improve *the quality of schema* via normalization, and to optimize queries and prevent invalid updates (see, *e.g.,* [1]). To *improve the quality of the data*, we need new forms of dependencies, such as conditional dependencies by specifying patterns of semantically related data values to capture data inconsistencies [23,7], matching dependencies by supporting similarity predicates to accommodate data errors in record matching [18], containment constraints by enforcing containment of certain information about core business entities in master data to reason about information completeness [20,19], and currency constraints by incorporating temporal orders to determine data currency [26].

Care must be taken when designing dependency languages for improving data quality. Among other things, we need to balance the tradeoff between expressive power and complexity, and revisit classical problems for dependencies such as the satisfiability, implication and finite axiomatizability analyses.

Improve Data Quality with Rules. After we come up with the "right" dependency languages for specifying data quality rules, the next question is how to effectively use these rules to improve data quality? In a nutshell, a rule-based data quality management system should provide the following functionality.

Discovering Data Quality Rules. To use dependencies as data quality rules, it is necessary to have efficient techniques in place that can *automatically discover* dependencies from data. Indeed, it is unrealistic to rely solely on human experts to design data quality rules via an expensive and long manual process, or count on business rules that have been accumulated. This suggests that we learn informative and interesting data quality rules from (possibly dirty) data, and prune away trivial and insignificant rules based on a threshold specified by users.

More specifically, given a database instance D, the *profiling problem* is to find a *minimal cover* of all dependencies (*e.g.,* CFDs, CINDs, matching dependencies) 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.

To find data quality rules, several algorithms have been developed for discovering CFDs [8,24,35] and matching dependencies [44].

Validating Data Quality Rules. A given set Σ 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 Σ, *i.e.,* those rules that make sense, to be used as data quality rules. Moreover, we need to

deduce new rules and to remove redundancies from Σ, via the implication or deduction analysis of those dependencies in Σ.

This problem is, however, nontrivial. It is already NP-complete to decide whether a given set of CFDs is satisfiable [23], and it becomes undecidable for CFDs and CINDs taken together [7]. Nevertheless, there has been an approximation algorithm for extracting a set S' of consistent rules from a set S of possibly inconsistent CFDs, while guaranteeing that S' is within a constant bound of the maximum consistent subset of S (see [23] for details).

Detecting Errors. After a validated set of data quality rules is identified, the next question concerns how to effectively catch errors in a database by using these 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. When it comes to relative information completeness, we want to decide whether D has complete information to answer an input query Q, among other things.

We have shown that for a centralized database D, given a set Σ of CFDs and CINDs, a fixed number of SQL queries can be *automatically* generated such that, when being evaluated against D, the queries return all and only those tuples in D that violate Σ [23]. That is, we can effectively detect inconsistencies by leveraging existing facility of commercial relational database systems.

In practice a database is often fragmented, vertically or horizontally, and is distributed across different sites. In this setting, inconsistency detection becomes nontrivial: it necessarily requires certain data to be shipped from one site to another. In this setting, error detection with minimum data shipment or minimum response time becomes NP-complete [25], and the SQL-based techniques for detecting violations of conditional dependencies no longer work. Nevertheless, effective batch algorithms [25] and incremental algorithms [29] have been developed for detecting errors in distributed data, with certain performance guarantees.

Data Imputation. After the errors are detected, we want to automatically localize the errors, fix the errors and make the data consistent. We also need to identify tuples that refer to the same entity, and for each entity, determine its latest and most accurate values from the data in our database. When some data are missing, we need to decide what data we should import and where to import from, so that we will have sufficient information for tasks at hand. As remarked earlier, these should be carried out by capitalizing on the interactions between processes for improving various aspects of data quality, as illustrated in Section 2.6.

As another example, let us consider *data repairing* for improving data consistency. 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 [2]. This is the method that US national statistical agencies, among others, have been practicing for decades for cleaning census data [33,36]. The data repairing problem is, nevertheless, highly nontrivial: it is NP-complete even when a fixed set of FDs or a fixed set of INDs is used as data quality rules [5], even for centralized databases. In light of these, several heuristic algorithms have been developed, to effectively repair data by employing FDs and INDs [5], CFDs [11,45], CFDs and matching

dependencies [27] as data quality rules. A functional prototype system [22] has also shown promises as an effective tool for repairing data in industry.

The data repairing methods mentioned above are essentially heuristic: while they improve the consistency of the data, they do not guarantee to find correct fixes for each error detected, *i.e.,* they do not warrant a precision and recall of 100%. Worse still, they may introduce new errors when trying to repair the data. In light of these, they are not accurate enough to repair critical data such as medical data, in which a minor error may have disastrous consequences. This highlights the quest for effective methods to find *certain fixes* that are guaranteed correct. Such a method has recently be proposed in [28]. While it may not be able to fix all the errors in our database, it guarantees that whenever it updates a data item, it correctly fixes an error without introducing any new error.

4 Conclusion

Data quality is widely perceived as one of the most important issues for information systems. In particular, the need for studying data quality is evident in big data management, for which two central issues of equivalent importance concern how to cope with *the quantity* of the data and *the quality* of the data.

The study of data quality management has raised as many questions as it has answered. It is a rich source of questions and vitality for database researchers. However, data quality research lags behind the demands in industry. A number of open questions need to be settled. Below we address some of the open issues.

Data Accuracy. Previous work on data quality has mostly focused on data consistency and data deduplication. In contrast, the study of data accuracy is still in its infancy. One of the most pressing issues concerns how to determine whether one value is more accurate than another in the absence of reference data. This calls for the development of models, quantitative metrics, and effective methods for determining the relative accuracy of data.

Information Completeness. Our understanding of this issue is still rudimentary. While the theory of relative information completeness [20,19] circumvents the limitations of the CWA and the OWA and allows us to determine whether a database has complete information to answer our query, effective metrics and algorithms are not yet in place for us to conduct the evaluation in practice.

Data Currency. The study of data currency has not yet reached the maturity. The results in this area are mostly theoretical: a model for specifying data currency, and complexity bounds for reasoning about the currency of data [26]. Among other things, effective methods for evaluating the currency of data in our databases and for deriving current values from stale data are yet to be developed.

Interaction between Various Issues of Data Quality. As remarked earlier, there is an intimate connection between data repairing and data deduplication [27]. Similarly, various interactions naturally arise when we attempt to

improve the five central aspects of data quality: information completeness is intimately related to data currency and consistency, and so is data currency to data consistency and accuracy. These interactions require a full treatment.

Repairing Distributed Data. Already hard to repair data in a centralized database, it is far more challenging to efficiently fix errors in distributed data. This is, however, a topic of great interest to the study of big data, which are typically partitioned and distributed. As remarked earlier, data quality is a central aspect of big data management, and hence, effective and scalable repairing methods for distributed data have to be studied.

The Quality of Complex Data. Data quality issues are on an even larger scale for data on the Web, *e.g.,* XML data and social graphs. Already hard for relational data, error detection and repairing are far more challenging for data with complex structures. 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 [30,32], let alone for data repairing. In this setting data quality remains by and large unexplored. Another 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, and spam detection, among other things.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Arenas, M., Bertossi, L.E., Chomicki, J.: Consistent query answers in inconsistent databases. In: PODS (1999)
3. Batini, C., Scannapieco, M.: Data Quality: Concepts, Methodologies and Techniques. Springer (2006)
4. Bertossi, L.: Database Repairing and Consistent Query Answering. Morgan & Claypool Publishers (2011)
5. Bohannon, P., Fan, W., Flaster, M., Rastogi, R.: A cost-based model and effective heuristic for repairing constraints by value modification. In: SIGMOD (2005)
6. Bravo, L., Fan, W., Geerts, F., Ma, S.: Increasing the expressivity of conditional functional dependencies without extra complexity. In: ICDE (2008)
7. Bravo, L., Fan, W., Ma, S.: Extending dependencies with conditions. In: VLDB (2007)
8. Chiang, F., Miller, R.: Discovering data quality rules. In: VLDB (2008)
9. 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)
10. Codd, E.F.: Relational completeness of data base sublanguages. In: Data Base Systems: Courant Computer Science Symposia Series 6. Prentice-Hall (1972)
11. Cong, G., Fan, W., Geerts, F., Jia, X., Ma, S.: Improving data quality: Consistency and accuracy. In: VLDB (2007)
12. Dong, X.L., Berti-Equille, L., Srivastava, D.: Integrating conflicting data: The role of source dependence. In: VLDB (2009)

13. Dong, X.L., Berti-Equille, L., Srivastava, D.: Truth discovery and copying detection in a dynamic world. In: VLDB (2009)
14. Eckerson, W.W.: Data quality and the bottom line: Achieving business success through a commitment to high quality data. The Data Warehousing Institute (2002)
15. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: A survey. TKDE 19(1) (2007)
16. English, L.: Plain English on data quality: Information quality management: The next frontier. DM Review Magazine (April 2000)
17. Fan, W.: Dependencies revisited for improving data quality. In: PODS (2008)
18. Fan, W., Gao, H., Jia, X., Li, J., Ma, S.: Dynamic constraints for record matching. VLDB J. 20(4), 495–520 (2011)
19. Fan, W., Geerts, F.: Capturing missing tuples and missing values. In: PODS, pp. 169–178 (2010)
20. Fan, W., Geerts, F.: Relative information completeness. TODS 35(4) (2010)
21. Fan, W., Geerts, F.: Foundations of Data Quality Management. Morgan & Claypool Publishers (2012)
22. Fan, W., Geerts, F., Jia, X.: Semandaq: A data quality system based on conditional functional dependencies. In: VLDB, demo (2008)
23. Fan, W., Geerts, F., Jia, X., Kementsietsidis, A.: Conditional functional dependencies for capturing data inconsistencies. TODS 33(1) (2008)
24. Fan, W., Geerts, F., Li, J., Xiong, M.: Discovering conditional functional dependencies. TKDE 23(5), 683–698 (2011)
25. Fan, W., Geerts, F., Ma, S., Müller, H.: Detecting inconsistencies in distributed data. In: ICDE, pp. 64–75 (2010)
26. Fan, W., Geerts, F., Wijsen, J.: Determining the currency of data. TODS (to appear)
27. Fan, W., Li, J., Ma, S., Tang, N., Yu, W.: Interaction between record matching and data repairing. In: SIGMOD (2011)
28. Fan, W., Li, J., Ma, S., Tang, N., Yu, W.: Towards certain fixes with editing rules and master data. VLDB J. 21(2), 213–238 (2012)
29. Fan, W., Li, J., Tang, N., Yu, W.: Incremental detection of inconsistencies in distributed data. In: ICDE (2012)
30. Fan, W., Libkin, L.: On XML integrity constraints in the presence of DTDs. J. ACM 49(3), 368–406 (2002)
31. Fan, W., Ma, S., Hu, Y., Liu, J., Wu, Y.: Propagating functional dependencies with conditions. In: VLDB, pp. 391–407 (2008)
32. Fan, W., Siméon, J.: Integrity constraints for XML. JCSS 66(1), 256–293 (2003)
33. Fellegi, I., Holt, D.: A systematic approach to automatic edit and imputation. J. American Statistical Association 71(353), 17–35 (1976)
34. Gartner. Forecast: Enterprise software markets, worldwide, 2008-2015, 2011 update. Technical report, Gartner (2011)
35. Golab, L., Karloff, H., Korn, F., Srivastava, D., Yu, B.: On generating near-optimal tableaux for conditional functional dependencies. In: VLDB (2008)
36. Herzog, T.N., Scheuren, F.J., Winkler, W.E.: Data Quality and Record Linkage Techniques. Springer (2009)
37. Loshin, D.: Master Data Management. Knowledge Integrity, Inc. (2009)
38. Miller, D.W., et al.: Missing prenatal records at a birth center: A communication problem quantified. In: AMIA Annu. Symp. Proc. (2005)
39. Naumann, F., Herschel, M.: An Introduction to Duplicate Detection. Morgan & Claypool Publishers (2010)

40. Otto, B., Weber, K.: From health checks to the seven sisters: The data quality journey at BT (September 2009), BT TR-BE HSG/CC CDQ/8
41. Redman, T.: The impact of poor data quality on the typical enterprise. Commun. ACM 2, 79–82 (1998)
42. SAS (2006), http://www.sas.com/industry/fsi/fraud/
43. Shilakes, C.C., Tylman, J.: Enterprise information portals. Technical report. Merrill Lynch, Inc., New York (November 1998)
44. Song, S., Chen, L.: Discovering matching dependencies. In: CIKM (2009)
45. Yakout, M., Elmagarmid, A.K., Neville, J., Ouzzani, M.: GDR: a system for guided data repair. In: SIGMOD (2010)
46. Zhang, H., Diao, Y., Immerman, N.: Recognizing patterns in streams with imprecise timestamps. In: VLDB (2010)