Foundations of Data Quality Management

Wenfei Fan
University of Edinburgh

Floris Geerts
University of Antwerp

SYNTHESIS LECTURES ON DATA MANAGEMENT #29

MORGAN & CLAYPOOL PUBLISHERS

ABSTRACT

Data quality is one of the most important problems in data management. A database system typically aims to support the creation, maintenance, and use of large amount of data, focusing on the quantity of data. However, real-life data are often dirty: inconsistent, duplicated, inaccurate, incomplete, or stale. Dirty data in a database routinely generate misleading or biased analytical results and decisions, and lead to loss of revenues, credibility and customers. With this comes the need for data quality management. In contrast to traditional data management tasks, data quality management enables the detection and correction of errors in the data, syntactic or semantic, in order to improve the quality of the data and hence, add value to business processes.

This monograph gives an overview of fundamental issues underlying central aspects of data quality, namely, data consistency, data deduplication, data accuracy, data currency, and information completeness. We promote a uniform logical framework for dealing with these issues, based on data quality rules. The text is organized into seven chapters, focusing on relational data. Chapter 1 introduces data quality issues. A conditional dependency theory is developed in Chapter 2, for capturing data inconsistencies. It is followed by practical techniques in Chapter 3 for discovering conditional dependencies, and for detecting inconsistencies and repairing data based on conditional dependencies. Matching dependencies are introduced in Chapter 4, as matching rules for data deduplication. A theory of relative information completeness is studied in Chapter 5, revising the classical Closed World Assumption and the Open World Assumption, to characterize incomplete information in the real world. A data currency model is presented in Chapter 6, to identify the current values of entities in a database and to answer queries with the current values, in the absence of reliable timestamps. Finally, interactions between these data quality issues are explored in Chapter 7. Important theoretical results and practical algorithms are covered, but formal proofs are omitted. The bibliographical notes contain pointers to papers in which the results were presented and proven, as well as references to materials for further reading.

This text is intended for a seminar course at the graduate level. It is also to serve as a useful resource for researchers and practitioners who are interested in the study of data quality. The fundamental research on data quality draws on several areas, including mathematical logic, computational complexity and database theory. It has raised as many questions as it has answered, and is a rich source of questions and vitality.

KEYWORDS

data quality, data consistency, data deduplication, data accuracy, information completeness, data currency, data dependencies, dependency discovery, rule validation, error detection, data repairing, master data, certain fixes

In memory of my mother (1933–2009)

Wenfei Fan

Contents

Acknowledgments

This book is an account of recent research and development in the area of data quality. Our first debt of gratitude is to Philip Bohannon, Loreto Bravo, Gao Cong, Michael Flaster, Xibei Jia, Anastasios Kementsietsidis, Laks V.S. Lakshmanan, Jianzhong Li, Shuai Ma, Heiko Müller, Rajeev Rastogi, Nan Tang, Jef Wijsen, Ming Xiong, and Wenyuan Yu, for working with us on the subject. The joint work with them has served as the basis of this book. We are also grateful to our colleagues at our home institutions for their unfailing support: the Database Group at the University of Edinburgh and the ADReM Research Group at the University of Antwerp.

We are grateful to Tamer Özsu for the thorough reading of the book and for his valuable comments. We would also like to thank Peter Buneman, Leopoldo Bertossi, Jan Chomicki, Ting Deng, and Scott Weinstein for their comments on a first version of this book.

Tamer Özsu and Diane Cerra guided us through the materialization and publication of this monograph. It was a pleasure to work on this project with them.

Part of the book was written when Wenfei was visiting Harbin and Shenzhen, China. He would like to thank Chong Chen and Huan Tu for the facilities and help they provided.

We would also like to thank EPSRC (029213/1), the 973 Program (2012CB316200), and NSFC (61133002) of China, the RSE-NSFC Joint Project Scheme, and IBM, for their support of our research on the subject.

Finally, Wenfei would like to thank his daughter, Grace (Nuonuo), for her patience, understanding, and love for all these years.

Wenfei Fan and Floris Geerts
July 2012

CHAPTER 1

Data Quality: An Overview

Database texts typically teach us how to design databases, formulate queries, and speed up query evaluation. We are led to believe that as long as we get a query right, then our database management system (DBMS) will find the correct answer to the query, and *voilà!* Unfortunately, this may not happen. In the real world, data are often dirty. Given dirty data in a database, we are not warranted to get accurate, complete, up-to-date or even correct answer to our query, no matter how well we write our query and how efficient our DBMS is. These highlight the need for improving *the quality of the data*.

This chapter identifies central issues in connection with data quality, and introduces a uniform logical framework to deal with these issues, based on data quality rules.

1.1 DATA QUALITY MANAGEMENT

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 1.1.

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). A DBMS will tell us that 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 in the U.S.; and similarly, when CC = 01 and AC = 908, city should be Murray Hill (MH) in the U.S. It is thus likely that NYC is not the true city value of t_1, t_2, and t_3. Second, the information 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.

	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

Figure 1.1: An employee instance.

(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 efficiently it processes our queries.

Unfortunately, real-life data are often *dirty*: inconsistent, duplicated, inaccurate, incomplete, and out of date. Indeed, enterprises typically find data error rates of approximately 1–5%, and for some companies it is above 30% [Redman, 1998]. In most data warehouse projects, data cleaning accounts for 30–80% of the development time and budget [Shilakes and Tylman, 1998], for improving the quality of the data rather than for 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" [Miller Jr. et al., 2005]. When data currency is concerned, it is known that "2% of records in a customer file become obsolete in one month" [Eckerson, 2002]. 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 U.S. businesses 600 billion dollars annually [Eckerson, 2002], and that erroneously priced data in retail databases alone cost U.S. consumers $2.5 billion each year [English, 2000]. While these indicate the daunting cost of dirty data in the U.S., 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.

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, timely, 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 [Gartner, 2011]. As an example, data quality tools deliver "an overall business value of more than 600 million GBP" each year at British Telecom [Otto and Weber, 2009]. Data quality management is also a critical part of big data management, master data management (MDM) [Loshin, 2009], customer relationship management (CRM), enterprise resource planning (ERP), and supply chain management (SCM), among other things.

1.2 CENTRAL ISSUES OF DATA QUALITY

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

1.2.1 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 relation (table), and between tuples across different relations.

As an example, consider tuples t_1, t_2, and t_3 in Figure 1.1. There are discrepancies and conflicts within each of these tuples, as well as inconsistencies between different tuples.

(1) It is known that in the UK (when CC = 44), if the area code is 131, then the city should be Edinburgh (EDI). In tuple t_1, however, CC = 44 and AC = 131, but 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 U.S. (CC = 01), if the area code is 908, the city should be Murray Hill (MH). Nevertheless, CC = 01 and 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 [Abiteboul et al., 1995]). As will be seen in Chapter 2, errors in a single relation can be detected by intrarelation constraints such as extensions of functional dependencies, while errors across different relations can be identified by interrelation constraints such as extensions of inclusion dependencies.

1.2.2 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, and object identification (for data with complex structures).

For example, consider tuples t_4, t_5, and t_6 in Figure 1.1. 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 that indicates that Mary Smith and Mary Luth have the same email account and hence, are the same person.

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 [SAS, 2006]. 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 U.S. Social Security Administration. The finding was quite surprising: there were 40 pilots whose records turned up in both databases (cf. [Herzog et al., 2009]).

No matter how important it is, data deduplication is nontrivial. As will be seen in Chapter 4, 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 are pairwise equal to each other. Worse still, it is often too costly to compare and examine every pair of tuples from large data sources.

1.2.3 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:

$$\mathsf{person(FN, LN, age, height, status)},$$

where a tuple specifies the name (FN, LN), age, height, and marital status of a person. A person instance is shown Figure 1.2.3, in which s_0 presents the "true" information for Mike. From these we can conclude that s_1[age, height] are more accurate than s_2[age, height] as they are closer to the true values of Mike, while s_2[FN, status] are more accurate than s_1[FN, status].

	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

Figure 1.2: A person instance.

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 the other 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[age] is more accurate than s_2[age]. That is, s_1[age] is closer to Mike's true age value than s_2[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[age]), we may deduce that s_2[status] may be more accurate than s_1[status].

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

1.2.4 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 the entities that the data in our database intend to represent, both attribute values and tuples may be missing from our databases. For instance, the value of t_1[phn] in the relation D_0 of Figure 1.1 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, for example, [Abiteboul et al., 1995]).

- The CWA assumes that a database has collected all the tuples representing real-world entities, but the *values* of some attributes in those tuples are possibly *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 those real-world entities.

Database theory is typically developed under the CWA. Unfortunately, in practice, one often finds that not only attribute values but also tuples are missing from our database. That is, the CWA often does not hold. On the other hand, under the OWA, we can expect few sensible queries to find complete answers.

As will be seen in Chapter 5, neither the CWA nor the OWA is quite appropriate in emerging applications such as master data management. In other words, databases in the real world are *neither* entirely closed-world *nor* entirely open-world. These databases are actually "partially closed." The good news is that we often find that partially closed databases have complete information to answer our queries at hand.

1.2.5 DATA CURRENCY

Data currency is also known as *timeliness*. It aims to identify the current values of entities represented by tuples in a database, 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 [Zhang et al., 2010]. Add to this the complication that data values are often copied or imported from other sources [Berti-Equille et al., 2009; Dong et al., 2010, 2009a,b], 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 1.1 given above. Assume that tuples t_4, t_5, and t_6 are found pertaining to the same employee Mary by using data deduplication techniques [Elmagarmid et al., 2007]. 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. As will be seen in Chapter 6, it is often possible to deduce currency orders from the semantics of the data, as illustrated below.

(1) While we do not have reliable timestamps associated with Mary's salary, we may 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]. That is, Mary's current last name is Luth.

	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

Figure 1.3: An example office master data relation.

1.2.6 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, these issues interact with each other, as illustrated below.

As we have observed earlier, tuples t_1, t_2, and t_3 in the relation D_0 of Figure 1.1 are inconsistent. We next 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 denoted by D_m and given in Figure 1.3. It is specified by schema:

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

As will be seen in Chapter 7, we may "repair" t_1, t_2, and t_3 as follows.

(1) If the values of attributes CC and AC of these tuples are confirmed accurate, we can safely update their city attributes by letting $t_1[\text{city}] = t_2[\text{city}] := \text{EDI}$, and $t_3[\text{city}] := \text{MH}$, 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 (street, city, zip), then the two tuples "match," i.e., they refer to the same address and phone. Hence, we can update t[CC, AC, phn] by taking the corresponding master values from t_m. This allows us to change $t_2'[\text{street}]$ to $t_{m1}[\text{street}]$. That is, we repair $t_2'[\text{street}]$ 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 $t_1'[\text{phn}]$ by letting $t_1'[\text{phn}] := t_2''[\text{phn}]$, 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 $t_2[\text{city}]$ is corrected. On the other hand, deduplication also helps us resolve conflicts: $t_1'[\text{phn}]$ is enriched only after $t_2'[\text{street}]$ 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 the relation D_0 of Figure 1.1, 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 often need 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 available data residing in our database.

These suggest that a practical data quality management system should provide functionality to deal with each and every one of the five central issues given above, and moreover, leverage the interactions between these issues to improve data quality.

1.3 IMPROVING DATA QUALITY USING RULES

We have seen that real-life data are often dirty, and that 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.

(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. 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. In this book 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 better still, fix those errors by using the rules. But what data quality rules should we adopt?

A natural idea is to use data dependencies (integrity constraints). Dependency theory is almost as old as relational databases themselves. Since Codd [1972] introduced functional dependencies, a variety of dependency languages, defined as various classes of first-order logic sentences, have been proposed and studied. 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, as will be seen later, all 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, for example, [Abiteboul et al., 1995]). To *improve the quality of the data*, we need new forms of dependencies, by specifying patterns of semantically related data values to capture data inconsistencies, supporting similarity predicates to accommodate data errors in data deduplication, enforcing the containment of certain information about core business entities in master data for reasoning about information completeness, and by incorporating temporal orders to determine data currency.

When developing dependencies for improving data quality, 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.

Improving 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 often unrealistic to rely solely on human experts to design data quality rules via an expensive and long manual process, and it is typically inadequate to 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 set by the users.

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 analysis of those dependencies in Σ.

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. We may also want to decide whether D has complete and current information to answer an input query Q, among other things.

Repairing data. After the errors are detected, we want to automatically localize the errors, fix the errors, and make the data consistent, as illustrated in Section 1.2.6. 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 attribute values or tuples are missing, we need to decide what data we should import and where to import from, so that we will have sufficient information for the tasks at hand. As remarked earlier, these should be carried out by exploring and capitalizing on the interactions between processes for improving various aspects of data quality.

1.4 BACKGROUND

We focus on the quality of relational data in this monograph. Data consistency will be covered in Chapters 2 and 3, followed by data deduplication, information completeness, and data currency in Chapters 4, 5, and 6, respectively. We study their interactions in Chapter 7.

We assume that the reader is familiar with the relational data model and the standard notions of schemas, instances, data dependencies and query languages (see [Abiteboul et al., 1995]). We also assume the knowledge of complexity theory (see, for example, [Papadimitriou, 1994]). In particular, we use the following notations.

(1) A database is specified by a relational schema $\mathcal{R}$, which consists of a collection of relation schemas $(R_1, \ldots, R_n)$. Each relation schema R_i is defined over a set of attributes, denoted by $\mathsf{attr}(R)$. For each attribute $A \in \mathsf{attr}(R)$, its domain is specified in R, denoted by $\mathsf{dom}(A)$. We use A, B, C and X_i, Y_i to range over attributes in $\mathsf{attr}(R)$, and W, X, Y, Z to range over sets (or lists) of attributes.

(2) We consider the following query languages (see [Abiteboul et al., 1995] for details):

- conjunctive queries (CQ), built up from atomic formulas with constants and variables, i.e., relation atoms in database schema $\mathcal{R}$ and built-in predicates $(=, \neq, <, \leq, >, \geq)$, by closing under *conjunction* $\wedge$ and *existential quantification* $\exists$;
- union of conjunctive queries (UCQ) of the form $Q_1 \cup \cdots \cup Q_r$, where for each $i \in [1, r]$, Q_i is in CQ;
- positive existential FO queries ($\exists$FO$^+$), built from atomic formulas by closing under $\wedge$, *disjunction* $\vee$ and $\exists$;
- first-order logic queries (FO) built from atomic formulas using $\wedge$, $\vee$, *negation* $\neg$, $\exists$ and *universal quantification* $\forall$; and
- datalog queries (FP), defined as a collection of rules $p(\bar{x}) \leftarrow p_1(\bar{x}_1), \ldots, p_n(\bar{x}_n)$, where each p_i is either an atomic formula (a relation atom in $\mathcal{R}$, $=$, $\neq$) or an IDB predicate. In other words, FP is an extension of $\exists$FO$^+$ with an *inflational fixpoint operator*.

BIBLIOGRAPHIC NOTES

Data quality has been a longstanding issue for decades, and the prevalent use of the Web has increased the risks, on an unprecedented scale, of creating and propagating dirty data. There have been several books on various topics in connection with data quality [Batini and Scannapieco, 2006; Bertossi, 2011; Herzog et al., 2009; Naumann and Herschel, 2010].

The need for studying the quality of relational schemas was already recognized when Codd [1970] introduced the relational model. Starting with keys [Codd, 1970], functional dependencies [Codd, 1972], and inclusion dependencies [Fagin, 1981], a variety of integrity constraints have been studied, including equality-generating dependencies and tuple-generating dependencies [Beeri and Vardi, 1984] (see [Abiteboul et al., 1995; Maier, 1983; Vardi, 1987] for a historical account and surveys).

To improve the quality of data, rather than schemas, Fellegi and Holt [1976] introduced the concept of *edits* to detect and repair inconsistent census data. The use of integrity constraints in improving the consistency of relational data was first formalized by Arenas et al. [1999], which introduced two approaches: *repair* is to find another database that is consistent and minimally differs from the original database, and *consistent query answer* is to find an answer to a given query in every repair of the original database. We consider data repairing in this book, and refer the reader to [Bertossi, 2011] for a comprehensive lecture on consistent query answering.

There has also been a host of work on data deduplication. We refer the interested reader to [Elmagarmid et al., 2007; Herzog et al., 2009; Naumann and Herschel, 2010].

The subject of information completeness has also received much attention; see, for example, [Abiteboul et al., 1995; Grahne, 1991; Imieliński and Lipski Jr, 1984; van der Meyden, 1998] for surveys.

The temporal database community has studied how to incorporate temporal information into the relational model, in terms of timestamps; see, for example, [Chomicki and Toman, 2005; Snodgrass, 1999; van der Meyden, 1997] for surveys.

The study of data accuracy is still in its infancy, and its formal treatment is not yet in place. This issue will be briefly discussed in Chapter 7.

Much more extensive bibliographic comments will be provided in the subsequent chapters.

Beyond the relational model, there has also been work on improving the quality of XML data (e.g., [Flesca et al., 2005; Weis and Naumann, 2005]) and the quality of results returned in searches (e.g., [Cafarella et al., 2009; Dong et al., 2009a; Galland et al., 2010; Yin et al., 2008]). These are beyond the scope of this book.

Authors' Biographies

WEIFEI FAN

Wenfei Fan is the (Chair) Professor of Web Data Management in the School of Informatics, University of Edinburgh, UK. He is a Fellow of the Royal Society of Edinburgh, UK, a National Professor of the 1000-Talent Program, and a Yangtze River Scholar, China. He received his Ph.D. from the University of Pennsylvania, U.S.A., and his M.S .and B.S. from Peking University, China. He is a recipient of the Alberto O. Mendelzon Test-of-Time Award of ACM PODS 2010, the Best Paper Award for VLDB 2010, the Roger Needham Award in 2008 (UK), the Best Paper Award for IEEE ICDE 2007, the Outstanding Overseas Young Scholar Award in 2003 (China), the Best Paper of the Year Award for Computer Networks in 2002, and the Career Award in 2001 (USA). His current research interests include database theory and systems, in particular data quality, data integration, database security, distributed query processing, query languages, social network analysis, Web services, and XML.

FLORIS GEERTS

Floris Geerts is Research Professor in the Department of Mathematics and Computer Science, University of Antwerp, Belgium. Before that, he held a senior research fellow position in the database group at the University of Edinburgh, UK and a postdoctoral research position in the data mining group at the University of Helsinki, Finland. He received his Ph.D. in 2001 from the University of Hasselt, Belgium. His research interests include the theory and practice of databases and the study of data quality, in particular. He is a recipient of the Best Paper Awards for IEEE ICDM 2001 and IEEE ICDE 2007.