
Data Mining and Knowledge Discovery, 7, 81–99, 2003

A Taxonomy of Dirty Data

WON KIM*
Cyber Database Solutions, Inc., Austin, Texas, USA

BYOUNG-JU CHOI†
Department of Computer Science, Ewha Institute of Science and Technology, Seoul, Korea

EUI-KYEONG HONG
University of Seoul, AITrc, Seoul, Korea

SOO-KYUNG KIM
Lucent Technologies, Seoul, Korea

DOHEON LEE
Department of Biosystems, Korea Advanced Institute of Science and Technology, Daejon, Korea

Editors: Fayyad, Mannila, Ramakrishnan

Received November 19, 2001; Revised November 20, 2001

Abstract. Today large corporations are constructing enterprise data warehouses from disparate data sources in order to run enterprise-wide data analysis applications, including decision support systems, multidimensional online analytical applications, data mining, and customer relationship management systems. A major problem that is only beginning to be recognized is that the data in data sources are often "dirty". Broadly, dirty data include missing data, wrong data, and non-standard representations of the same data. The results of analyzing a database/data warehouse of dirty data can be damaging and at best be unreliable. In this paper, a comprehensive classification of dirty data is developed for use as a framework for understanding how dirty data arise, manifest themselves, and may be cleansed to ensure proper construction of data warehouses and accurate data analysis. The impact of dirty data on data mining is also explored.

Keywords: dirty data, data quality, data mining, data cleansing, data warehousing

1. Introduction

Today, data warehousing systems are becoming a key enabler of corporate information technology infrastructure. Corporations have recognized the value of data at their disposal as an important asset that can make them more competitive in today's dynamic business environment. By consolidating data from disparate data sources into a "central" data warehouse, corporations may be able to run data analysis applications and obtain information

*This research was partially supported by Korea's Brain Korea-21 grant.
†This research was partially supported by Korea's KISTEP grant.

that is of strategic and tactical importance for their businesses (TechGuide-1; Ballou and Tayi, 1999; Inmon, 1999). Data warehouses are being constructed in various industries, such as telecommunications, financial services, insurance, retail, healthcare, etc. There are scores of software products that assist in the creation of data warehouses (Golfarelli and Rizzi, 1999; Inmon, 1996; Kimball et al., 1998), analysis of data (Berson and Smith, 1997), mining of data (Berry and Linoff, 1997; Westphal and Blaxton, 1998), and customer relationship management (CRM) (Applied Technology, 1998; First Logic Inc.; TechGuide-2; IBM, NUMA-Q, 1999).

These applications are based on the use of business intelligence derived from data warehouses or databases, and underscore the importance of high quality data. Data quality has been a subject of longstanding discussions (English, 1999; Wang et al., 1995), and there are even software products that help cleanse dirty data (Vality Technology Inc.; Trillium; Trillium, 1998; Williams, 1997) on the market. However, only now it is beginning to be recognized that an inordinate proportion of data in most data sources is "dirty". Roughly speaking, dirty data means either missing data or wrong data or non-standard representations of the same data (Williams, 1997; Cutter Information Corporation, 1998). Before data analysis applications are applied against any data, the data must be cleansed to remove or repair dirty data. Further, data from legacy data sources (e.g., mainframe-based COBOL programs) do not even have metadata that describe them. To the best of our knowledge, there is no comprehensive formal taxonomy of dirty data or a metric for data quality. Without such a taxonomy or metric, it will remain difficult to know with a high degree of confidence the quality of business intelligence derived from data warehouses and the quality of decisions made on the basis of such business intelligence.

One primary objective and contribution of this paper is to develop a comprehensive taxonomy of dirty data. The taxonomy provides a framework for understanding the origins of a complete spectrum of dirty data and the impact of dirty data on data mining, and sheds light on techniques for dealing with dirty data and for defining a metric for measuring data quality. We expect that such a taxonomy will provide a valuable guideline for further research and enhancement of commercial products.

For the purpose of this paper, we define dirty data and sources of dirty data as follows.

- The life cycle of data includes its capture, storage, update, transmission, access, archive, restore, deletion, and purge. The focus of our research is on the access aspect by a user or application that operates correctly. As such, we say that data is dirty if the user or application ends up with a wrong result or is not able to derive a result due to certain inherent problems with the data.
- The sources of dirty data include data entry error by a human or computer system, data update error by a human or computer system, data transmission error by a computer system, and even bugs in a data processing computer system.

We limit the scope of the paper with the following assumptions.

- Access to stored data is accomplished by presenting a sample data in a query condition. We assume the sample data will be in the same national language and notational standards as those used for the stored data. For example, we assume that the user or application will not look for French date notation in an English date field.

- The types of data considered in this paper are only numerical and string data. In particular, we exclude multimedia data from consideration. Multimedia data, such as images, audio, and video, require rather different considerations, as the types of access to multimedia data are rather different from those to alphanumeric data.
- In this paper, we consider only dirty "data", not metadata. One of the authors of this paper already provided a taxonomy of semantic heterogeneity in metadata that arises when integrating different, independently created databases (Kim and Seo, 1991; Kim et al., 1993).

2. Taxonomy of dirty data

Table 1 summarizes our taxonomy of dirty data. In this section, we describe the taxonomy and present a case for "near-completeness" of the taxonomy by describing the logic behind the structure of the taxonomy. As we proceed we illustrate each category of dirty data with examples as appropriate.

In order to arrive at a "comprehensive" taxonomy, we adopt the standard "successive hierarchical refinement" approach. The key is to keep the fan-out factor small (2 or 3) wherever possible, at each non-leaf node of the taxonomy hierarchy, such that it would be intuitively obvious that there are no other meaningful child nodes of any given node.

We note that our taxonomy is based on the premise that dirty data manifests itself in three different ways: missing data, not missing but wrong data, and not missing and not wrong but unusable. The third way occurs when two or more databases are integrated or representation standards are not consistently used in inputting data. The taxonomy is a hierarchical decomposition of these three basic manifestations of dirty data. As such, each successive level of the hierarchy represents a form of manifestation of dirty data. We note that some dirty data manifests itself as a combination of more than one type of dirty data (e.g., a concatenated data in wrong ordering and with misspelling—"Kenedy, John", instead of "John Kennedy"), but our taxonomy includes only "primitive" types of dirty data, and not any "composite" types of dirty data. Our taxonomy consists of 33 leaf-level, or primitive, dirty data types. We note that, although some of the leaf-level nodes could be further decomposed into "finer types" of dirty data, we have chosen not to do so, as such an exercise will yield only marginal additional insight into understanding dirty data.

As we will show in the next section, different taxonomies will result if we start from different premises. However, the set of dirty data types in each taxonomy will be the same. We also note that we are only confident that our taxonomy may be about 95% (i.e., very close, but not quite) "comprehensive". (We will make clear the reason for our hedging later in this section.) However, the fact that our taxonomy may not be 100% "comprehensive" does not diminish its significance and usefulness. (This will become clear in Section 3.)

Now let us examine the structure of the taxonomy in some detail. The root node of the taxonomy has only two child nodes: missing data (1) and not-missing data (2). Obviously, at this point, the taxonomy is complete, since there cannot be a third child node. Missing data is data that is missing (in a field) when it should not be missing. Not-missing data is data that is entered, properly or otherwise, in a field.

Table 1. Taxonomy of dirty data.

1. Missing data
   - 1.1 Missing data where there is no Null-not-allowed constraint
   - 1.2 Missing data where Null-not-allowed constraint should be enforced
2. Not-missing, but
   - 2.1 Wrong data, due to
     - 2.1.1 Non-enforcement of automatically enforceable integrity constraints
       - 2.1.1.1 Integrity constraints supported in relational database systems today
         - 2.1.1.1.1 User-specificable constraints
           - 2.1.1.1.1.1 Use of wrong data type (violating data type constraint, including value range)
           - 2.1.1.1.1.2 Dangling data (violating referential integrity)
           - 2.1.1.1.1.3 Duplicated data (violating non-null uniqueness constraint)
           - 2.1.1.1.1.4 Mutually inconsistent data (action not triggered upon a condition taking place)
         - 2.1.1.1.2 Integrity guaranteed through transaction management
           - 2.1.1.1.2.1 Lost update (due to lack of concurrency control)
           - 2.1.1.1.2.2 Dirty read (due to lack of concurrency control)
           - 2.1.1.1.2.3 Unrepeatable read (due to lack of concurrency control)
           - 2.1.1.1.2.4 Lost transaction (due to lack of proper crash recovery)
       - 2.1.1.2 Integrity constraints not supported in relational database systems today
         - 2.1.1.2.1 Wrong categorical data (e.g., wrong abstraction level, out of category range data)
         - 2.1.1.2.2 Outdated temporal data (violating temporal valid time constraint; e.g., a person's age or salary not having been updated)
         - 2.1.1.2.3 Inconsistent spatial data (violating spatial constraint; e.g., incomplete shape)
     - 2.1.2 Non-enforceability of integrity constraints
       - 2.1.2.1 Data entry error involving a single table/file
         - 2.1.2.1.1 Data entry error involving a single field
           - 2.1.2.1.1.1 Erroneous entry (e.g., age mistyped as 26 instead of 25)
           - 2.1.2.1.1.2 Misspelling (e.g., principle instead of principal, effect instead of affect)
           - 2.1.2.1.1.3 Extraneous data (e.g., name and title, instead of just the name)
         - 2.1.2.1.2 Data entry error involving multiple fields
           - 2.1.2.1.2.1 Entry into wrong fields (e.g., address in the name field)
           - 2.1.2.1.2.2 Wrong derived-field data (due to error in functions for computing data in a derived field)
       - 2.1.2.2 Inconsistency across multiple tables/files (e.g., the number of Employees in the Employee table and the number of employees in the department table do not match)
   - 2.2 Not wrong, but unusable data
     - 2.2.1 Different data for the same entity across multiple databases (e.g., different salary data for the same person in two different tables or two different databases)
     - 2.2.2 Ambiguous data, due to
       - 2.2.2.1 Use of abbreviation (Dr. for doctor or drive)
       - 2.2.2.2 Incomplete context (homonyms; and Miami, of Ohio or Florida)
     - 2.2.3 Non-standard conforming data, due to
       - 2.2.3.1 Different representations of non-compound data
         - 2.2.3.1.1 Algorithmic transformation is not possible
           - 2.2.3.1.1.1 Abbreviation (ste for suite, hwy for highway)

(*Continued on next page*.)

Table 1. (*Continued*).

2.2.3.1.1.2	Alias/nick name (e.g., Mopac, Loop 1, and Highway 1; Bill Clinton, President Clinton, William Jefferson Clinton)
2.2.3.1.2	Algorithmic transformation is possible
2.2.3.1.2.1	Encoding formats (ASCII, EBCDIC, ...)
2.2.3.1.2.2	Representations (including negative number, currency, date, time, precision, fraction)
2.2.3.1.2.3	Measurement units (including date, time, currency, distance, weight, area, volume, ...)
2.2.3.2	Different representations of compound data
2.2.3.2.1	Concatenated data
2.2.3.2.1.1	Abbreviated version (e.g. John Kennedy for John Fitzgerald Kennedy)
2.2.3.2.1.2	Uses of special characters (space, no space, dash, parenthesis, in a social security number or phone number)
2.2.3.2.1.3	Different orderings (John Kennedy vs. Kennedy, John)
2.2.3.2.2	Hierarchical data (e.g. address concept hierarchy: state-county-city vs. state-city)
2.2.3.2.2.1	Abbreviated version
2.2.3.2.2.2	Uses of special characters
2.2.3.2.2.3	Different orderings (city-state, state-city)

The missing data node (1) is divided into (1.1) missing data due to the data being unknown or being "don't care" (when Null data is allowed), and (1.2) missing data despite the fact that missing data entry (i.e., Null data) is not allowed. It is clear that, with respect to the Null data being allowed or not, there cannot be a third child node. Missing data (1.1) is known as Null data (Date, 2000). In this case, Null data is not dirty data. However, when the data becomes known, the Null data must be replaced with the known correct data. If such replacement is not done, the data becomes dirty data. An example of category (1.1) missing data is an "Employee's supervisor" being missing (due to its not being known) within an Employee record during the initial phase of an Employee's employment. An example of category (1.2) missing data may be the "identification number" of an Employee, which is mandatory for any Employee.

The problems of Null data have been addressed in the literature in the context of relational databases. Codd (1979) proposed a three-valued logic to deal with uncertainty in relations and incorporating nulls in the relational algebra to address the missing information problem. Dey and Sarkar (1996) proposed a "probabilistic relational model", an approach to uncertainty in data values based on probability theory instead of nulls and three-valued logic. Date (1998) described a systematic approach to the missing information problem that is based on special values and two-valued logic instead of nulls and three-valued logic.

The not-missing data node (2) is split into two child nodes: wrong (and so unusable) data (2.1) and not-wrong but unusable data (2.2). It is clear that there cannot be a third child node. A wrong (and so unusable) data is data that is different from the "true value" of the data at the time the data is accessed. A not-wrong but unusable data is data that is in some sense not wrong, but can result in wrong results in a query or analysis. Examples of wrong data include the use of a character string in a field whose required data type is integer, 225 for an Employee's age, 25 as an Employee's age when in the same record the Employee's year

of birth is entered as 1980 (i.e., the true age of the Employee is 20), misspelling "President Clinton" as "Persident Clinton", etc. Examples of not wrong but unusable data include the use of a city name "Miami" without specifying its State (Miami is a city in both the State of Florida and State of Ohio), the use of an abbreviation "ste" instead of "suite", the use of different representations of date (April 15, 4/15, 04/15), etc.

The not-wrong but unusable (2.2) data is dirty data that arises due to differences between data maintained in more than two independent databases or due to incomplete or non-standard specification of data in one database. For example, John Smith's salary in one database is 40000, while it is 20000 in another database. Each data may be correct, as John Smith keeps two jobs. However, when the two databases are integrated, this will cause confusion. For example, also, if the address of a company is stored in a record as "ste. 256", but if the search condition in a query includes "suite *", the query will not match the stored record. Similarly, if a query searches for stored records with "April 15" in the date field by using a search condition "4/15", the records with "April 15" will not be found.

Wrong data (2.1) branches into two child nodes: wrong data that can be prevented through automatic enforcement of integrity constraints (2.1.1), and wrong data that cannot be prevented through automatic enforcement of integrity constraints (2.1.2). Obviously, with respect to preventing wrong data through automatic enforcement of integrity constraints, there cannot be a third child node.

Wrong data that can be prevented through use of automatically enforceable integrity constraints (2.1.1) in turn is split into two child nodes in terms of whether such constraints are supported in current relational database systems (2.1.1.1) or they require, theoretically feasible, extensions to today's relational database systems (2.1.1.2). There cannot be a third child node with respect to whether certain database integrity constraints are supported or not supported in today's relational database systems.

We note that although it is not our intention by any means to limit the scope of dirty data to those that arise when using relational database systems, we have chosen to distinguish wrong data in terms of whether they can or cannot be prevented by the techniques supported in today's relational database systems. This is simply because today's relational database systems provide mechanisms to prevent 9 types of wrong data from corrupting a database. This has been well established in the database field during the past three decades. We use this fact to establish "completeness" of one major category of dirty data, namely 2.1.1.1.

Broadly, there are two types of mechanism for enforcing database integrity in today's relational database systems. They are user-specified integrity constraints ((2.1.1.1.1) and (1.2)) (Silberschatz et al., 1997) and transaction management (2.1.1.1.2) (Traiger et al., 1982; Gray and Reuter, 1993). User-specified integrity constraints include a data type (or domain) constraint on each field (2.1.1.1.1.1), referential integrity (or foreign key-primary key) constraint (2.1.1.1.1.2), uniqueness constraint (2.1.1.1.1.3), triggers (2.1.1.1.1.4) and Null-not-allowed constraint (1.2). We note that the users (application developers) must specify these constraints, and database systems automatically enforce them, only when they have been specified. Database systems have no way of knowing what integrity constraints to enforce, since they do not know the semantics of data.

The data type constraint enforces the type of data (and even the length and precision of data), but not the contents of the data, that can be entered in a field. For example, if the data

type constraint that the data type of the Employee's age field is integer, the database system will prevent string data from being entered into the field. However, the database system is not able to determine if a particular Employee's age is 26 or 25, or even whether 225 is a valid age for an Employee. A special data type is the "value range" data type (e.g., integer 18..65 for Employee's age) that can be used to some extent to control the contents of data. The data type constraints enforced in today's relational database systems operate on string data, Boolean data, and continuous numerical data. In other words, support for data type constraints on categorical data is weak (we will discuss this shortly below).

The referential integrity constraint guarantees the existence of a logical linkage between data in one table and data in another table, and so prevents a dangling reference. A dangling reference arises when there is a data in one table with no counterpart in another table; for example, when there is no Department name in the Department table when there is a reference to that department name in the Department field of the Employee table. The uniqueness constraint guarantees that every data in a given field (or combination of fields) is unique and non-Null, and is imposed on "key" fields (e.g., social security number of Employees).

A wide variety of mutually inconsistent data (2.1.1.1.1.4) can be prevented through use of triggers. A trigger is a rule of the basic form <IF condition THEN action>. The "condition" may be any Boolean expression, and the "action" is any action that a database system can perform. For example, the (IF Employee.age > 69 THEN delete Employee) trigger will result in the deletion of an Employee record when the Employee age becomes greater than 69. The trigger is a powerful mechanism that is more general than for enforcing integrity constraints on data in single records. Because of the generality of the user-specifiable "action" (and the "condition"), the trigger can be especially powerful in enforcing integrity constraints that span multiple tables/files. For example, IF Employee's job is updated to "manager" THEN insert a new record in the Department table or update the manager field in the Department table for the Employee's Department, and send a promotion-announcement email to all Employees.

Transaction management facilities (2.1.1.1.2) in today's relational database systems prevent four other types of wrong data: through concurrency control to prevent "lost update" (2.1.1.1.2.1), "dirty read"(2.1.1.1.2.2), and "unrepeatable read" (2.1.1.1.2.3); and through recovery to prevent lost transaction (2.1.1.1.2.4). We note that transaction management facilities guarantee that the four types of dirty data do not occur as long as the computer system that manages the data is not destroyed. This is a strong guarantee indeed.

When two or more transactions read and update the same data simultaneously, two types of anomaly may result. A lost update occurs when, for example, transaction T1 reads the "number of seats available in a flight" as 1, assigns it to a customer and decrements the available seat count to 0, while transaction T2 reads the same data at the same time, assigns it to another customer and decrements the available seat count to 0. In this case, one of the two updates has been lost. A dirty data read occurs when, for example, transaction T1 increments the available seat count, due to a cancellation, from 2 to 3, then transaction T2 reads the updated available seat count and assigns the 3 seats to 3 customers, and then transaction T1 aborts (thereby undoing the first update). In this case, transaction T2 has read "dirty data" (this is a term used in the context of transaction management and refers to "uncommitted"

data within a transaction) written by transaction T1. When transaction T1 reads "the number of available seats in a flight" and finds it to be 5, and transaction T2 updates the seat count to 10 to reflect a cancellation of five seat reservations. If transaction T1 reads the seat count again and finds the count to be 10, the read is said to be unrepeatable. Unrepeatable reads are undesirable, since the different reads means "dirty data" (uncommitted data) that is liable to change again. A "lost transaction" occurs when, for example, in a fund transfer transaction, a debit of $200 is made against a savings account, and before the $200 is credited to a checking account, the transaction or the computer system crashes. If the system cannot properly recover from the crash, the $200 from the savings account will have evaporated. The "atomicity and durability" properties of transactions supported by transaction management facilities (either in relational database systems or transaction processing monitors (Gray and Reuter, 1993)) guarantee that all updates within a transaction are either committed or backed out as a single unit (atomic), and that once a transaction has committed, the effects are permanent (durable).

Wrong data that arise due to non-enforcement of integrity constraints that are not supported in today's relational database systems but that can theoretically be supported, with extensions to today's systems, (2.1.1.2) branch into three child nodes. These include integrity constraints that are possible on categorical data (2.1.1.2.1), temporal data (2.1.1.2.2), and spatial data (2.1.1.2.3). This view is well established in the database field. Despite three decades of research into temporal data (time point, time interval, time attribute hierarchy) (Snodgrass, 1995; Etzion et al., 1998) and spatial data (point, line, polygon) (Ooi, 1990; Laurini and Thompson, 1993; Schneider, 1997), today's relational database systems only support a bare minimum of capabilities for these types of data. The need to support categorical data (50 US States; income status in terms of "super-rich, rich, middle-income, poor, poverty-stricken") has recently been accentuated due to limitations in input data types that data mining algorithms can accept (Stokes et al., 1995; Berry and Linoff, 1997; Berson and Smith, 1997).

Examples of invalid categorical data include a category that is not one of the valid user-specified categories. We note that we include use of wrong abstraction levels (e.g., "frozen food" or "frozen pizza" instead of "food") as a type of wrong categorical data. One may make a case that all descendants of a node in an abstraction hierarchy (is-a-kind-of hierarchy or generalization hierarchy) are related by shared semantics, and so may be used interchangeably. However, the trouble is that today there is no database system (relational, object-oriented, or object-relational) that supports "instance-level generalization" queries, that is, retrieve any or all descendants of an instance (object). For example, it is not possible to request retrieval of any or all descendants of a "food" object from any of today's database system. Support of generalization hierarchy in object-oriented and object-relational database systems applies only to metadata (i.e., type hierarchy), not to individual instances (objects). Therefore, if "frozen food" is entered, instead of "food", a query looking for "food" will not find "frozen food"; and vice versa. Recall that we defined dirty data at the outset as data that causes an application to end up with no result or wrong result. As such, we have chosen to include wrong abstraction levels as wrong data.

The temporal data constraint specifies the time instant or time interval during which a data is valid (e.g., an Employee's salary entered in a field is no longer valid when the Employee's

salary is raised). The spatial data constraint specifies the spatial relationships that must be satisfied (e.g. the point coordinates should combine to yield a closed rectangle). The spatial constraint may involve data across multiple fields within a record, since the coordinate data may be specified in a combination of fields, rather than in a single field. It may be that future versions of object-relational database systems (Kim, 1995; Stonebraker, 1996) will provide native support for enforcing constraints on temporal and spatial data, treating them as abstract data types.

We note that one may make a case for including what we classify as "different representations of non-compound data" (2.2.3.1) and "different representations of compound data" (2.2.3.2) as child nodes of (2.1.1.2). If one takes the view that different representations of the same data can be prevented if a standard representation is specified and enforced as a form of integrity constraint, then these types of dirty data may be regarded as wrong data. However, if one takes the view that even if a single standard representation is enforced, if more than one database is to be integrated, the differences will cause a conflict that must be resolved. Then these different representations of the same data are indeed not wrong data, but simply unusable until a single standard is to be adopted and non-conformed representations are brought to conformance. We take the latter view.

Wrong data that cannot be prevented through use of automatically enforceable integrity constraints (2.1.2) is beyond control by today's or near-term future database technology. This category includes situations where it is practically impossible to even specify integrity constraints. For example, how would one go about preventing a person from misspelling "principal" (as principle), "effect" (as affect), "Deng Xiao-Ping" (as Dong Shaw Ping), etc.? Also, how would a data processing system know, without some type of cross-reference checks, that an Employee's age has been correctly entered, even if there is a data value range constraint imposed on the age field? In any case, this type of wrong data is divided into wrong data occurring within a single table or file (2.1.2.1), and one occurring across multiple tables or files (2.1.2.2). With respect to whether a wrong error occurs within a single table or multiple tables, it is clear that there is no third alternative.

Wrong data within a single table (2.1.2.1) is split into two child nodes: wrong data due to data entry error involving a single field (2.1.2.1.1) and data entry error arising from inconsistency among data in more than one field (2.1.2.1.2). Again, with respect to the number of fields involved, there cannot be a third child node of (2.1.2.1).

We decompose (2.1.2.1.1) into three types of wrong data (2.1.2.1.1.1 through 2.1.2.1.1.3). Although we have not been able to come up with additional child nodes, given the "creative" way in which people can make data entry errors (Vality Technology Inc.) and the subtle semantics of data, we suspect that a few additional types of wrong data may be possible under (2.1.2.1.1). One possible child node is wrong data due to inconsistency among data in multiple fields. For example, if the "age" field has 25, and the "birth year" field has 1980, at least one of the data is wrong. Wrong data involving spatial data, as we have seen above, tends to fall into this type. However, because of the existence of wrong data type (2.1.1.1.1.4 mutually inconsistent data), we have chosen not to create a new child node of (2.1.2.1.1).

An example of an erroneous entry (2.1.2.1.1.1) is 26 for an Employee's age, rather than 25, due to a slippery finger. Wrong data due to misspelling (2.1.2.1.1.2) is obvious. An

example of an extraneous data (2.1.2.1.1.3) is the entry of a name and title (John Williams, President and CEO) in the "name" field.

The wrong data due to data entry error involving multiple fields (2.1.2.1.2) is split into two nodes: entry into wrong fields, and wrong derived-field data from stored data. Here again, we suspect that a few additional types of wrong data may be possible under (2.1.2.1.2). An example of an entry into wrong fields (2.1.2.1.2.1) is the entry of a street address in the "name" field. The wrong data due to wrong derived-field data (2.1.2.1.2.2) arises due to errors in computing data for a derived field. Examples include a miscalculation of an Employee's net income by mis-computing the tax; and mis-concatenation of the street address, county, city, and state in a wrong order.

Wrong data that manifests as different data for the same real-world entity or concept across multiple tables/files (2.1.2.2) arises because integrity constraints that encompass all semantically related tables are not specified and enforced. An example is the situation where the number of Employees that is obtained by counting the number of records in the Employee table is different from the number of Employees that is obtained by summing the number of Employees in each Department in the Department table. Many wrong data of this type can be controlled by use of the triggers. But there are situations that may be difficult to control using triggers, especially when the condition that activates the action part of the trigger reads dirty data (e.g., misspelled string, string with incomplete context).

The not-wrong but unusable data (2.2) is decomposed into three child nodes: unusable due to differences across multiple databases (2.2.1), unusable due to ambiguity (2.2.2), and unusable due to non-conformance to standards (2.2.3). We have not been able to come up with a fourth possibility. As we observed earlier, the not-wrong but unusable data may be regarded as wrong data if such data has been entered despite the presence of constraints or policies for adopting a single standard representation. However, if more than one independent database is to be integrated, they become unusable in the context of the integrated database, even if they are each correct. One may point out that the same situation arises if, for example, data has been entered in different databases using different data types, different integrity constraints, different number of fields to represent certain data (e.g., Employee's home address), etc. This is a valid view, of course. However, this is an issue of schema (metadata)-level heterogeneity, and is outside the scope of this paper. We refer readers interested in classifying and neutralizing schema-level heterogeneity to Kim and Seo (1991) and Kim et al. (1993).

As an example of different data for the same entity (2.2.1), the "salary" data for an Employee may be entered in one database as "54000", while for the same Employee it may appear in another database as "48000". The difference may be due to the fact that the Employee has two jobs, and both "salary" data are correct.

The ambiguous data (2.2.2) is decomposed into two child nodes: ambiguity due to use of abbreviation (2.2.2.1) and ambiguity due to incomplete context (2.2.2.2). Again, we have not been able to come up with another possibility. An example of ambiguous data due to use of abbreviation is "MS", which may stand for "Microsoft", "MicroStrategy", "Morgan Stanley", etc. (as a name of a corporation). Examples of ambiguous data due to incomplete context include the aforementioned city name "Miami", which may be in the State of Florida or the State of Ohio; and homonyms hot (temperature) and hot (spice), pool

(billiard) and (swimming) pool, etc. Certain homonyms are likely to be inserted, correctly, in the same field (of different records), and can lead to erroneous answers to certain types of query.

Unusable data due to non-conformance to standards (2.2.3) is divided into two child nodes: different representations of non-compound data (2.2.3.1), and different representations of compound data (2.2.3.2). A third child node is not possible.

We note that there have been various proposals for representing incomplete data in the context of relational databases. One is to allow set-valued attributes, along with extended semantics of relational operators (Buckles and Petry, 1982). Others attempted to incorporate possibility distributions, fuzzy sets and rough sets into relational databases (Zemenkova and Kandel, 1985; Galindo et al., 2001; Sozat and Yazici, 2001). Recently, the use of an information-theoretic connectionist network has been proposed to discover unreliable data in a relational database (Maimon et al., 2001).

Different representations of non-compound data (2.2.3.1) branches into two child nodes: one for which algorithmic transformation is not possible (2.2.3.1.1), and one for which algorithmic transformation is possible (2.2.3.1.2). A third child node is not possible. Different representations of non-compound data for which algorithmic transformations are not possible (2.2.3.1.1) includes two child nodes: use of abbreviation (2.2.3.1.1.1) and use of alias/nick name (2.2.3.1.1.2). One representation may be mapped to a standard representation only via a mapping table, address directory, etc., and there may be multiple equivalent representations for each standard representation. Examples of abbreviation (2.2.3.1.1.1) are "hwy" for "highway", and "ste" for "suite". Examples of aliases or nicknames (2.2.3.1.1.2) are "Mopac", "Loop 1" and "Highway 1" for the same highway in Austin, Texas; President Clinton, Bill Clinton, and William Jefferson Clinton for the same person.

Different representations of non-compound data for which algorithmic transformations are possible (2.2.3.1.2) includes three child nodes: encoding formats (2.2.3.1.2.1), representations (2.2.3.1.2.2), and measurement units (2.2.3.1.2.3). We have not been able to come up with another possibility. Examples of encoding formats are ASCII and EBCDIC; and male and female in the sex field of an Employee into m and f. Examples of representations are for negative number (−250 or (250)), currency (−$250, ($250), −250.39, (250.39)), date (April 15, 4/15), time (1:25:30, 85:30), precision (single vs. double precision), fraction (quarter, eighth)—that is, at least those supported in Microsoft Excel spreadsheet. Examples of measurements are for date (in units of 100 days), time (in units of 15 minutes), currency (in units of thousand dollars), distance (yard vs. meter), weight (pound vs. kilogram), area (square feet vs. square meters), volume (gallon vs. liter), etc.

Different representations of compound data (2.2.3.2) is split into two child nodes: concatenated data (2.2.3.2.1) and hierarchical data (2.2.3.2.2). A concatenated compound data is data that consists of two or more elements, such as a person's name (first name, middle initial, last name) or street address (apartment number, number, street). Ordering among the elements of a concatenated data is important. However, there is no conceptual hierarchy implied among the elements; that is, the last name does not "include" the first name, or vice versa. A hierarchical compound data is basically a concatenated data in which there is a concept hierarchy implied among some of the elements of the data. With respect to the structure of a compound data, it is clear that a third child node is not possible.

We note that non-conformance to standards by compound data may come in a variety of ways. Accordingly, (2.2.3.2.1) and (2.2.3.2.2) are each split into three child nodes: abbreviated version, uses of special characters, and different orderings. We have not been able to come up with a fourth possibility. An example of the use of abbreviated versions in a concatenated data is the use of a person's name without the middle name, such as John Kennedy rather than John Fitzgerald Kennedy. An example of the use of special (delimiting) characters in a concatenated data is the representation of the telephone number (512-249-9759 vs. 5122499759 vs. (512) 249-9759). An example of the use of different orderings in a concatenated data is (John Kennedy vs. Kennedy, John). An example of the use of an abbreviated hierarchical compound data is an address hierarchy, street address-city-state, rather than the full hierarchy of street address-city-county-state-zip code. An example of the use of special characters in a hierarchical compound data is (Texas, Williamson, Austin vs. Texas, (Williamson), Austin). An example of the use of different orderings for a hierarchical compound data is (Texas, Williamson, Austin vs. Austin, Williamson, Texas).

3. Taxonomy of techniques for dealing with dirty data

The taxonomy of dirty data that we developed in Section 2 immediately becomes of significant value. Table 2 shows the dirty data taxonomy of Table 1 with summaries of techniques for preventing, checking or repairing each type of dirty data. A sad conclusion that one can readily draw from an examination of Table 2 is that commercial data quality tools today do not address many types of dirty data, and most types of dirty data require "eyeballing", that is, examinations and repairs by humans with domain expertise for data analysis applications.

We note that of the 33 leaf-level dirty data types, at least 25 of them require intervention by a domain expert today. Of the 33 leaf-level dirty data types, only 9 can be prevented automatically (1.2, 2.1.1.1.1.1 through 2.1.1.1.1.4, and 2.1.1.1.2.1 through 2.1.1.1.2.4); and 8 of the 9 require the users to specify the constraints (the only exception being 2.1.1.1.2.4 prevention of lost transactions).

Various commercial software tools for creating data warehouses or transforming data for multidimensional analysis or data mining provide several ways to replace missing data (1.1) (SAS Institute Inc., 1999). They typically allow the users to replace missing data in a field with a mean value (arithmetic average), median value (the 50th percentile), midrange (average of a range between the maximum and minimum values), etc.

Commercial data quality tools, such as Trillium, First Logic and Vality, have been engineered over many years and are proving highly helpful in converting names and addresses in several countries into their standard and complete representations, with the aid of country-wide directories of names and addresses (Vality Technology Inc.; First Logic Inc.; Trillium). For example, these tools can even detect and correct wrongly entered street addresses. However, their usefulness is limited largely to helping detect erroneous and non-standard forms of names and addresses. In our taxonomy, they are applicable to only about 11 leaf-level dirty data types: (2.1.2.1.1.2—misspelling), (2.2.2.1—ambiguity due to use of abbreviation), (2.2.2.2—ambiguity due to incomplete context), (2.2.3.1.1.1—non-standard conformance by use of abbreviation), (2.2.3.1.1.2—non-standard conformance by use of aliases/nicknames), (2.2.3.2.1.1 through 2.2.3.2.1.3—concatenated compound data), and

Table 2. Taxonomy of dirty data for dealing with dirty data.

1. Missing data
   - 1.1 Missing data where there is no Null-not-allowed constraint
     [filling in a representative data (including Null), or *intervention by a domain expert*]
   - 1.2 Missing data where Null-not-allowed constraint should be enforced
     [disallow non-entry]
2. Not-missing, but
   - 2.1 Wrong data, due to
     - 2.1.1 Non-enforcement of automatically enforceable integrity constraints
       - 2.1.1.1 Integrity constraints supported in relational database systems today
         - 2.1.1.1.1 User-specificable constraints
           - 2.1.1.1.1.1 Use of wrong data type
             [type checking on data entry, or data profiling with *intervention by a domain expert*]
           - 2.1.1.1.1.2 Dangling data
             [referential integrity constraint, or data profiling with *intervention by a domain expert*]
           - 2.1.1.1.1.3 Duplicated data
             [uniqueness integrity constraint, or data profiling with *intervention by a domain expert*]
           - 2.1.1.1.1.4 Mutually inconsistent data
             [set triggers, or *intervention by a domain expert*]
         - 2.1.1.1.2 Integrity guaranteed through transaction management
           - 2.1.1.1.2.1 Lost update
             [2-phase locking and share/exclusive locking]
           - 2.1.1.1.2.2 Dirty read
             [2-phase locking and share/exclusive locking]
           - 2.1.1.1.2.3 Unrepeatable read
             [2-phase locking and share/exclusive locking]
           - 2.1.1.1.2.4 Lost transaction
             [transaction management facility with log-based recovery]
       - 2.1.1.2 Integrity constraints not supported in relational database systems today
         - 2.1.1.2.1 Wrong categorical data
           [use of a category lookup table or a pre-built computerized abstraction hierarchy, or *intervention by a domain expert*]
         - 2.1.1.2.2 Outdated temporal data
           [method for enforcing a temporal constraint, *or intervention by a domain expert*]
         - 2.1.1.2.3 Inconsistent spatial data
           [method for enforcing a spatial constraint, *or intervention by a domain expert*]
     - 2.1.2 Non-enforceability of integrity constraints
       - 2.1.2.1 Data entry error involving a single table/file
         - 2.1.2.1.1 Data entry error involving a single field
           - 2.1.2.1.1.1 Erroneous entry
             [*intervention by a domain expert*]
           - 2.1.2.1.1.2 Misspelling
             [run a spell-checker, and/or *intervention by a domain expert* or just someone who can spell]
           - 2.1.2.1.1.3 Extraneous data
             [*intervention by a domain expert* or someone who can edit]
         - 2.1.2.1.2 Data entry error involving multiple fields

(*Continued on next page*.)

Table 2. (*Continued*).

- 2.1.2.1.2.1 Entry into wrong fields
  [*intervention by a domain expert* or someone who can edit]
- 2.1.2.1.2.2 Wrong derived-field data
  [*intervention by a domain expert*, fix the function and reapply]
- 2.1.2.2 Inconsistency across multiple tables/files
  [*intervention by a domain expert*]

2.2 Not wrong, but unusable data

- 2.2.1 Different data for the same entity across multiple databases
  [*intervention by a domain expert*]
- 2.2.2 Ambiguous data, due to
  - 2.2.2.1 Use of abbreviation
    [use of a name and address lookup table, use of an abbreviation dictionary, and/or *intervention by a domain expert*]
  - 2.2.2.2 Incomplete context
    [use of a name and address lookup table, use of a thesaurus, and/or *intervention by the domain expert*]
- 2.2.3 Non-standard conforming data, due to
  - 2.2.3.1 Different representations of non-compound data
    - 2.2.3.1.1 Algorithmic transformation is not possible
      - 2.2.3.1.1.1 Abbreviation
        [use of a name and address lookup table, use of an abbreviation dictionary, and/or *intervention by a domain expert*]
      - 2.2.3.1.1.2 Alias/nick name
        [use of a name and address lookup table, and/or *intervention by a domain expert*]
    - 2.2.3.1.2 Algorithmic transformation is possible
      - 2.2.3.1.2.1 Encoding formats
        [use of encoding tables, conversion algorithms/software]
      - 2.2.3.1.2.2 Representations
        [use of conversion algorithms/software]
      - 2.2.3.1.2.3 Measurement units
        [use of conversion algorithms/software]
  - 2.2.3.2 Different representations of compound data
    - 2.2.3.2.1 Concatenated data
      - 2.2.3.2.1.1 Abbreviated version
        [use of lookup tables such as a zip-code and phone directory, and/or *intervention by a domain expert*]
      - 2.2.3.2.1.2 Uses of special characters
        [use of lookup tables such as a zip-code and phone directory, and/or *intervention by a domain expert*]
      - 2.2.3.2.1.3 Different orderings
        [use of lookup tables such as a zip-code and phone directory, and/or *intervention by a domain expert*]
    - 2.2.3.2.2 Hierarchical data
      - 2.2.3.2.2.1 Abbreviated version
        [use of lookup tables such as a nationwide address directory, and/or *intervention by a domain expert*]
      - 2.2.3.2.2.2 Uses of special characters
        [use of lookup tables such as a zip-code and phone directory, and/or *intervention by a domain expert*]
      - 2.2.3.2.2.3 Different orderings
        [use of lookup tables such as a zip-code and phone directory, and/or *intervention by a domain expert*]

(2.2.3.3.2.1 through 2.2.3.2.2.3—hierarchical compound data). They are not even capable of dealing with all or most situations in any of these 11 dirty data types.

Five wrong data types can be prevented by having user-specified integrity constraints enforced by database and transaction processing systems, as we have shown in the previous section. These are (1.2—missing data where Null is not allowed), (2.1.1.1.1.1—wrong data type for a field), (2.1.1.1.1.2—dangling reference where it is not allowed), (2.1.1.1.1.3—duplicated data where duplicate is not allowed), and (2.1.1.1.1.4—complex constraints enforced through the trigger). When these types of wrong data entered into the database, and there is no metadata that describes such constraints, a technique known as "data profiling" (Olson) can be used to deduce such constraints (i.e., metadata) from the data. Due to the presence of dirty data or missing data, data profiling can deduce only the statistical likelihood of the constraints. Data profiling in general requires repeated scanning of all records in a table, and is highly time-consuming. There are some data profiling tools on the market, such as one from Evoke Software (Olson).

Four wrong data types, (2.1.1.1.2.1), (2.1.1.1.2.2), (2.1.1.1.2.3) and (2.1.1.1.2.4), can be prevented by utilizing transaction management facilities that are provided in a database system or transaction processing monitors. Transaction management facilities include a two-phase locking mechanism, multiple levels of isolation (of a transaction from other concurrent transactions), and log-based recovery mechanism.

Integrity constraints may be specified on categorical, temporal, and spatial data in user-defined methods for use in preventing or repairing wrong data types (2.1.1.2.1), (2.1.1.2.2) and (2.1.1.2.3), respectively. Object-oriented database systems or object-relational database systems provide the infrastructure for supporting such user-defined methods, but not relational database systems.

Most of the wrong data types (2.1.2) can only be repaired or prevented manually or semi-manually by human experts. A few of them can be repaired or prevented through use of tools. For example, wrong data due to misspelling (2.1.2.1.1.2) can be checked by running a spell checker; but spell checkers do not detect all typographical mistakes and often cannot detect mistakes, for example, in people's names and addresses.

The use of a name and address directory, a telephone directory, conversion tables, and a thesaurus are useful to a good extent for repairing or checking not-wrong but unusable data (2.2). However, they cannot be used to repair or check all types of unusable data, and domain experts need to step in. Typically, domain experts check and repair unusable data using these means as tools.

4. Impact of dirty data on data mining

If a "high" proportion of the dataset on which a data mining algorithm runs is dirty, obviously one cannot expect accurate results. Dirty data has varying degrees of adverse impact on all data mining algorithms. However, it is difficult to quantify or definitively characterize the impact, because of the statistical nature of the computations performed by the algorithms, data transformations that some of the algorithms require, different tolerances for noise (dirty or exceptional data) by different algorithms, and the nature of the applications for which the algorithms are run.

For the purpose of understanding the impact of dirty data on data mining algorithms, dirty data of all types may be reduced simply to wrong data, which in turn may be classified into wrong numerical data, wrong string data, and missing data. Certain data mining algorithms, such as association rules, can use numerical or string data "as is", that is, without requiring it to be first transformed into a different format. However, data mining algorithms such as the neural networks require data of any type to be transformed to numerical data between 0 and 1. Such algorithms as decisions trees require data to be transformed to categorical data. Missing data may simply be excluded from computation or filled in with data automatically generated by the data-cleansing component of a data mining algorithm.

A wrong numerical data (10,000, instead of 1,000) or a wrong string data ("New York", instead of "New Jersey") or a missing data, when used without transformation, is likely to contribution to producing an unreliable result, depending on the proportion of wrong or missing data relative to the entire dataset. A wrong numerical data or a wrong string data, when transformed to a categorical data or a numerical data between 0 and 1, is also likely to fall into a wrong category or a wrong numeric representation. A missing data, whether it is excluded in a computation or filled in with some "representative" data, is also likely to contribute to an incorrect result. Clearly, if the proportion of dirty data that results in wrong transformation or the proportion of missing data is "high", relative to the entire dataset, the results of data mining are likely to be unreliable. However, the magnitude of error relative to data transformation also matters. For example, a numeric data representing salary, 75,500 instead of 75,400, when converted to a categorical data in granules of 10,000, will not impact the result; but 100,000, instead of 10,000 will.

The impact of dirty data also depends on data mining algorithms. Certain data mining algorithms, such as decisions trees, neural networks, and Bayesian networks, require training (and testing, and evaluation). The presence of a high proportion of dirty data in the training dataset and/or the testing dataset is likely to render the resulting model less than reliable. If the dataset is not to be properly cleansed before being used for training and testing a model, at least a larger dataset should be used to reduce the impact of dirty data. Decision trees are known to be susceptible to noises, particularly if the trees are of higher order than two (binary trees) (Berry and Linoff, 1997). Neural networks are also known to be susceptible to noises. Bayesian networks are relatively less sensitive to noises arising from missing data by filling it using sampling and distribution methods. Certain data mining algorithms such as memory-based reasoning (K-nearest neighbor algorithm) and automatic cluster analysis (K-means algorithm) require formulation and use of distance functions, measures of association and similarity. The distance functions and measures of association and similarity are computed on the basis of data in the dataset. If the data used in computing these is dirty, they in turn are faulty, and the results of the algorithms become unreliable.

Even for the same data mining algorithm, the impact of dirty data depends on the application. For example, an application that aims to discover certain broad patterns is more tolerant of dirty data than one that aims to discover certain infrequently occurring patterns. In certain situations, a small number of outlying data is of primary interest (e.g., detecting failure situations involving automobile parts). In such situations, a "very low" proportion of dirty data contributes directly to wrong results.

5. Concluding remarks

In this paper, we developed what we believe is a very nearly comprehensive taxonomy of dirty data and explored the impact of dirty data on data mining results. The motivation for the research is the emergence of business intelligence systems, such as customer relationship management systems, decision support systems, multidimensional data analysis systems (or online analytical processing systems), and data mining systems. These systems are all designed to access a database or a data warehouse and extract stored data via standard queries, derive summary data from various perspectives on certain measure data (e.g., sales revenue, sales cost, unit sales), or extract unforeseen patterns.

Corporations have been pre-occupied until recently in building data warehouses or data marts (departmental data warehouses), and applying data analysis applications on them. However, the value of clean or quality data for use by such applications is only now receiving focused attention. Today, to the best of our knowledge, no comprehensive taxonomy of dirty data exists. Our objective was to put forth a taxonomy of dirty data, such that it will provide a basis for systematically understanding coverage of the taxonomy with available technologies for cleansing dirty data, and for establishing a metric for quantifying data quality in large and complex data sets.

By adopting the standard successive hierarchical refinement methodology, we tried to make a case for very near "comprehensiveness" of our dirty data taxonomy. After developing the taxonomy, we did a systematic analysis of the "coverage" of the taxonomy with available technologies for preventing and repairing dirty data, and found that today's technologies do not even address a half of the dirty data types we have been able to establish in our taxonomy.

The challenge, for both researchers, vendors of data quality products, and consumers and gatherers of data sets, is now clear. New techniques for significantly increasing the coverage of the dirty data taxonomy must be developed. Metrics for quantifying data quality must be developed for use in measuring the quality of data in data sets, and for guiding the collection and cleansing of data sets.

Acknowledgments

We thank the anonymous referees, Heikki Mannila, Raghu Ramakrishnan, and Jung-Won Lee (Ewha University, Korea) for their helpful comments. Their comments helped to improve the contents of the paper.

References

The Applied Technology Group. 1998. Building a successful CRM environment. White Paper, The Applied Technology Group, available at http://www.techguide.com/.

Ballou, D. and Tayi, G.K. 1999. Enhancing data quality in data warehouse environments. Communications of the ACM, 42(1):73–78.

Berry, M. and Linoff, G. 1997. Data Mining Techniques for Marketing, Sales and Customer Support. New York: John Wiley and Sons.

Berson, A. and Smith, S. 1997. Data Warehousing, Data Mining, and OLAP (Data Warehousing/Data Management). Computing, McGraw-Hill.

Buckles, B. and Petry, E. 1982. A fuzzy representation of data for relational databases. Fuzzy Sets and Systems, 7:213–226.

Codd, E.F. 1979. Extending the database relational model to capture more meaning. ACM Transaction on Database Systems, 4(4).

Cutter Information Corporation. 1998. Data management strategies newsletter on the state of the data warehousing industry. Management Science, 31:150–162.

Date, C. 1998. Faults and defaults. In Relational Database Writing 1994–1997 (in five parts). C.J. Date, H. Darwen, and D. McGoveran (Eds.). Reading, MA: Addison-Wesley.

Date, C. 2000. An Introduction to Database Systems, 7th edn. Reading, MA: Addison-Wesley.

Dey, D. and Sarkar, S. 1996. A probabilistic relational model and algebra. ACM Transactions on Database Systems, 21(3).

English, L. 1999. Improving Data Warehouse and Business Information Quality-Method for Reducing Costs and Increasing Profits. New York: Wiley.

Etzion, O., Jajodia, S., and Sripada, S. (Eds.). 1998. Temporal Databases: Research and Practice, Lecture Notes in Computer Science, Vol. 1399. Berlin: Springer-Verlag.

First Logic Inc. Customer data quality—Building the foundation for a one-to-one customer relationship. White Paper, available at http://www.firstlogic.com/.

Galindo, J., Medina, J.M., and Aranda-Garrido, M. 2001. Fuzzy division in fuzzy relational databases: An approach. Fuzzy Sets and Systems, 121:471–490.

Golfarelli, M. and Rizzi, S. 1999. Designing the data warehouse: Key steps and crucial issues. Journal of Computer Science and Information Management, 2(3).

Gray, J. and Reuter, A. 1993. Transaction Processing: Concepts and Techniques. San Mateo, CA: Morgan Kaufmann.

IBM NUMA-Q. 1999. Modeling customer relationship. White Paper, available at http://www.sequent.com/solutions/crm/whitepapers/mcr_wp.html.

Inmon, W.H. 1996. Building the Data Warehouse. New York: John Wiley.

Inmon, W.H. 1999. Data Warehouse Performance. New York: John Wiley.

Kim, W. and Seo, J.Y. 1991. On classifying schematic and data heterogeneity in multidatabase systems. IEEE Computer, 24(12).

Kim, W., Choi, I.J., Gala, S., and Scheevel, M. 1993. On resolving schema heterogeneity in multidatabase systems. Distributed and Parallel Databases, 1(1):251–279.

Kim, W. 1995. Modern Database Systems. ACM Press, 1995.

Kim, W., Chae, K.J., Cho, D.S., Choi, B.J., Kim, M., Lee, K.H., Lee, M.J., Lee, S.H., Park, S.S., and Yong, H.S. 1999. A component-based knowledge engineering architecture. Journal of Object-Oriented Programming, 12(6):40–48.

Kimball, R. et al. 1998. The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing, and Deploying Data Warehouses. New York: John Wiley.

Laurini, R. and Thompson, D. 1993. Fundamentals of Spatial Information Systems, A.P.I.C. Series no. 37. San Diego, CA: Academic Press.

Maimon, O., Kandel, A., and Last, M. 2001. Information-theoretic fuzzy approach to data reliability and data mining. Fuzzy Sets and Systems, 117:183–194.

Olson, J. Data profiling. White Paper, Evoke Software Corporation, available at http://www.evokesoft.com/products/ProdWPDP.html.

Ooi, B. 1990. Efficient Query Processing in Geographic Information Systems, Lecture Notes in Computer Science. Berlin: Springer-Verlag.

SAS Institute Inc. 1999. Finding the solution to data mining—A map of the features and components of SAS enterprise miner software version 3. White Paper, available at http://www.sas.com.

Schneider, M. 1997. Spatial Data Types for Database Systems: Finite Resolution Geometry for Geographic Information Systems, Lecture Notes in Computer Science, Vol. 1288. Berlin: Springer-Verlag.

Silberschatz, A., Korth, H., and Sudarchan, S. 1997. Database System Concepts. New York: McGraw-Hill.

Snodgrass, R. (Ed). 1995. The TSQL2 Temporal Query Language. Boston, MA: Kluwer Academic Publishers.

Sozat, M.I. and Yazici, A. 2001. A complete axiomatization for fuzzy functional and multivalued dependencies in fuzzy database relations. Fuzzy Sets and Systems, 117:161–181.

Stokes, M.E., Davis, C.S., and Koch, G.G. 1995. Categorical Data Analysis Using the SAS System. SAS Institute.

Stonebraker, M. 1996. Object-Relational DBMSs: The Next Great Wave. San Mateo, CA: Morgan Kaufmann Publishers.

TechGuide-1. The Technology Guide Series. A practical guide to achieving enterprise data quality—Trillium software. White Paper, available at http://www.techguide.com/.

TechGuide-2. The Technology Guide Series. Achieving business success through customer relationship management (CRM)—Mosaix. White Paper, available at http://www.techguide.com/.

Traiger, I., Gray, J., Galtieri, C.A., and Lindsay, B. 1982. Transactions and consistency in distributed database systems. ACM Trans. Database Systems, 7(3):323–342.

Trillium Software User Manual.

Trillium Software System. 1998. A practical guide to achieving enterprise data quality. White Paper, available at http://www.trilliumsoft.com/.

Vality Technology Inc. The five legacy data contaminants you will encounter in your warehouse migration. White Paper, available at http://www.vality.com/.

Wang, R., Storey, V., and Firth, C. 1995. A framework for analysis of data quality research. IEEE Transactions on Knowledge and Engineering, 7(4):623–640.

Westphal, C. and Blaxton, T. 1998. Data Mining Solutions: Methods and Tools for Solving Real-World Problems. New York: John Wiley.

Williams, J. 1997. Tools for traveling data. In DBMS. Miller Freeman.

Zemankova, M. and Kandel, A. 1985. Implementing imprecision in information systems. Information Sciences, 37:107–141.

Won Kim is President and CEO of Cyber Database Solutions, Inc. (www.cyberdb.com), in Austin, Texas. He is also Dean of Ewha Institute of Science and Technology at Ewha Women's University of Korea. He received a Ph.D. degree in Computer Science from the University of Illinois at Urbana-Champaign. His business and research interests include data warehousing, data mining, Web technologies, and multimedia contents processing.

Byoung-Ju Choi is an Associate Professor with the Department of Computer Science and Engineering at Ewha Women's University in Korea. She received a B.S. degree in Mathematics from Ewha Women's University and M.S. and Ph.D. degrees in Computer Science from Purdue University, USA. Her research interests are in software engineering with particular emphasis on software testing, data quality, and software process improvement.

Eui-Kyeong Hong is a Professor with the Department of Computer Science and Statistics, University of Seoul, Korea. He received a B.S. degree in Mathematics Education from Seoul National University, Korea, and M.S. and Ph.D. degrees in Computer Science from Korea Advanced Institute of Science and Technology (KAIST). His research interests include performance evaluation, distributed databases, data mining, and XML.

Soo-Kyung Kim is a Technical Associate at Lucent Technologies Korea, Bell Labs. She received B.S. and M.S. degrees in Computer Science from Ewha Women's University.

Doheon Lee is an Associate Professor with the Department of Biosystems at Korea Advanced Institute of Science and Technology (KAIST). He received B.S., M.S., and Ph.D. degrees from the Department of Computer Science at KAIST. His research interests include bioinformatics, data mining, database systems and information retrieval.