WORKING PAPER
ALFRED P. SLOAN SCHOOL OF MANAGEMENT

Quality Data Objects

December 1992

WP #3517-93
CISL WP# 92-06

Richard Y. Wang
M. P. Reddy

Sloan School of Management, MIT

MASSACHUSETTS
INSTITUTE OF TECHNOLOGY
50 MEMORIAL DRIVE
CAMBRIDGE, MASSACHUSETTS 02139

Quality Data Objects

December 1992

WP #3517-93
CISL WP# 92-06

Richard Y. Wang
M. P. Reddy

Sloan School of Management, MIT

* see page bottom for complete address

Richard Y. Wang E53-322
M. Prabhakara Reddy

Sloan School of Management
Massachusetts Institute of Technology
Cambridge, MA 01239

Quality Data Objects

Richard Y. Wang
M. Prabhakara Reddy

December 1992

CISL-92-06

Composite Information Systems Laboratory
E53-320, Sloan School of Management
Massachusetts Institute of Technology
Cambridge, Mass. 02139
ATTN: Prof. Richard Wang
(617) 253-0442
Bitnet Address: rwang@eagle.mit.edu

Acknowledgments Work reported herein has been supported, in part, by MIT's Total Data Quality Management (TDQM) project, MIT's Productivity From Information Technology (PROFIT) Consortium, MIT's International Financial Service Research Center (IFSRC) and MIT's Center for Information Systems Research (CISR). In particular, the authors wish to thank Prof. Stuart Madnick and Dr. Amar Gupta for their support to this research.

Quality Data Objects

ABSTRACT Needs for a quality perspective in the management of data resources are becoming increasingly critical. This paper investigates how to associate data with quality information that can help users make judgments of the quality of data. Specifically, we propose the concept of quality data object and investigate its structure and behavior. The structure of the quality data object includes a description of the datum object, its corresponding quality description object, and a mechanism to associate the datum object with its quality description object. The behavior of the quality object includes a set of methods to measure quality dimensions (such as timeliness, completeness, credibility). In addition, we have developed a quality data object algebra that includes quality comparison methods and an algebra that extends the relational algebra to the quality data object domain. It allows for a systematic construction of retrieval methods for quality data objects.

The concept of quality data object presented in the paper is a first step toward the design and manufacture of data products. We envision that the quality data object proposed in this paper can be used as basic building blocks for the design, manufacture, and delivery of quality data products. It will enable users to measure the quality of data products according to their chosen criteria; it will also enable users to buy data products based on their quality requirements. In this manner, we hope that the concepts of quality data objects and quality data products will help improve data quality and data reusability.

Quality Data Objects

1. Introduction

The quality of data in a database implemented by a conventional data base management systems (DBMS) has been treated, primarily, through functionalities such as recovery, concurrency, integrity, and security control (e.g., Bernstein & Goodman, 1981; Bernstein, et al., 1981; Codd, 1970; Codd, 1982; Codd, 1986; Date, 1981; Date, 1990; Denning & Denning, 1979; Fernandez, Summers, & Wood, 1981; Hoffman, 1977; Hsiao, Kerr, & Madnick, 1978; Korth & Silberschatz, 1986; Martin, 1973; Qian & Wiederhold, 1986; Ullman, 1982). Recovery restores the database to a state that is known to be correct after some failure has rendered the current state incorrect. Concurrency control ensures that the consistency of data is preserved when multiple users update the database concurrently. Integrity aims at preventing invalid updates against the database from happening. Invalid updates may be caused by errors in data entry, by mistakes on the part of the operator or the application programmer, by system failures, even by deliberate falsification; the last of these, however, is not so much a matter of integrity as it is of security; protecting the database against illegal operations, as opposed to those that are merely invalid, is the responsibility of the security subsystem (Date, 1985).

These functionalities are necessary but not sufficient to ensure data quality in the DBMS from the end-user's perspective (Johnson, Leitch, & Neter, 1981; Laudon, 1986; Liepins & Uppuluri, 1990; Liepins, 1989; Wang & Kon, 1992; Zarkovich, 1966). Integrity constraints and validity checks, for example, are essential to ensuring data quality in a database, but they are often just the beginning of a continuing data-integrity program that will ultimately address the real needs of users for data that can be used as an input to the user's decision making process (Maxwell, 1989). In general, data in the DBMS may be used by a range of different organizational functions with different perceptions of what constitutes quality data in terms of dimensions such as accuracy, completeness, consistency, and timeliness (Ballou & Pazer, 1987; Huh, et al., 1990; Redman, 1992).

Consider the following example scenarios:

- A person's name is carried as J. F. Rockart in once place, John F. Rockart in another, and Jack Rockart in yet another. All are technically "true" and would pass the integrity constraints provided by the conventional DBMS, but which one should be considered as <u>accurate</u> and stored in the database <u>consistently</u>?

- A client workstation runs business applications using data downloaded from a database server at the end of each day. Whereas, data in the server is updated instantly with changes and new information through on-line transaction processing. Thus, data in the client workstation is never current from the server and some user's viewpoint.
- Earning estimates for companies are stored in a database but who made these estimates, when, and how are not, making it difficult to judge the credibility of the data by those who are not familiar with the context.

In these and other similar situations, the quality of data managed by the DBMS is not so much a matter of data validity but rather of its usage. It would be useful to associate data with quality information that can help users make judgments of the quality of data for the specific application at hand. The research question here is how to structure and manage data in such a way that users can be equipped with the capabilities to measure the quality of data they need and to retrieve the data that conforms with their quality requirements.

1.1 Related work

An attribute-based research that facilitates cell-level tagging of data has been proposed to enable users to retrieve data that conforms with their quality requirements (Wang, Kon, & Madnick, 1993; Wang, Reddy, & Kon, 1992; Wang & Madnick, 1990). Included in this attribute-based research effort are a methodology for analyzing data quality requirements that extends the ER model proposed by Chen (Chen, 1976; Chen, 1984; Chen, 1991; Chen & Li, 1987), an attribute-based model encompassing a model description, a set of quality integrity rules, and a quality indicator algebra that extends the relational model proposed by Codd (Codd, 1970; Codd, 1979; Codd, 1982; Codd, 1986). The quality indicator algebra can be used to process SQL queries that are augmented with quality indicator requirements. From these quality indicators, the user can make a better judgment of the quality of data. The problem with this research is twofold: (1) In order to associate the application data with its corresponding quality description through the *join* operation in the model, an artificial link needs to be created through the concept of quality key. (2) In order to be able to judge the quality of data, it is necessary to compute data quality dimension values and other procedure-oriented quality measures. Although these could be accomplished using the relational approach, it is not as natural compared to that of the object-oriented approach. Moreover, this research did not address issues involved in measuring data quality dimension values.

In other related research efforts that aim at annotating data, self-describing data files and meta-data management have been proposed at the schema level (McCarthy, 1982; McCarthy, 1984;

McCarthy, 1988); however, no specific solution has been offered to manipulate such quality information at the instance level. In (Sciore, 1991), annotations are used to support the temporal dimension of data in an object-oriented environment. However, data quality is a multi-dimensional concept. Therefore, a more general treatment is necessary to address the data quality issue. More importantly, no algebra or calculus-based language is provided to support the manipulation of annotations associated with the data. Still other research efforts (Codd, 1979; Siegel & Madnick, 1991) have dealt with data tagging without either an algebra or a set of quality measures for data quality dimensions.

1.2. Research Focus

In this paper, we advocate that data quality must be modeled as an integral part of a data object rather than simply as a set of functionalities of the DBMS. More specifically, we propose the modeling construct of quality data object in which each datum is associated with appropriate data and procedures used to indicate the quality of the data object. We present a set of quality measure methods that compute quality dimension values (such as accuracy, consistency, completeness, and timeliness), and a set of quality algebraic methods that supports the manipulation of quality data objects.

Many concepts in the object-oriented paradigm can be applied to support the quality data object (Banerjee, 1987; Snyder, 1986). They are fundamental in our decision to model the quality data object via the object-oriented approach. The reader is referred to the Appendix for a detailed discussion of how constructs in the object-oriented paradigm such as inheritance, method, polymorphism, active value, and message can be applied to support the quality data object.

In this research, each datum is modeled as an object called a datum object. As shown in Figure 1, the quality information corresponding to the datum is called a quality description object. The *is-a-quality-of* link associates a quality description object with its datum object. The composite object constructed from a datum object and its associated quality description object is called a quality data object. Instance variables of a quality description object include descriptive data ($quality\_indicator_i$, i= 1, ..., n) and procedural data ($quality\_procedure_j$, j= 1, ..., m).

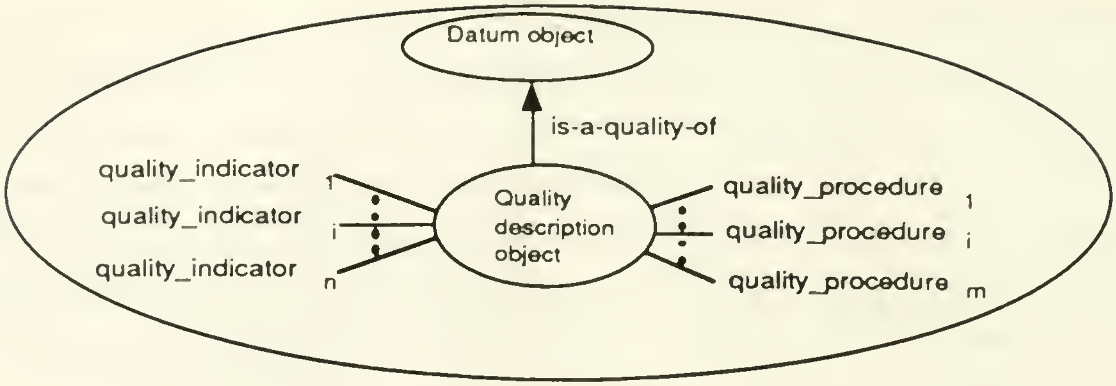

Figure 1: A Quality Data Object

A quality data object called Earnings-Estimate is exemplified in Figure 2. Note that Earnings-Estimate-Qual and Source-1-Qual are quality description objects for Earnings-Estimate and Source-1 respectively. Note also that Source-1, an attribute of the quality description object Earnings-Estimate-Qual, is itself a quality data object.

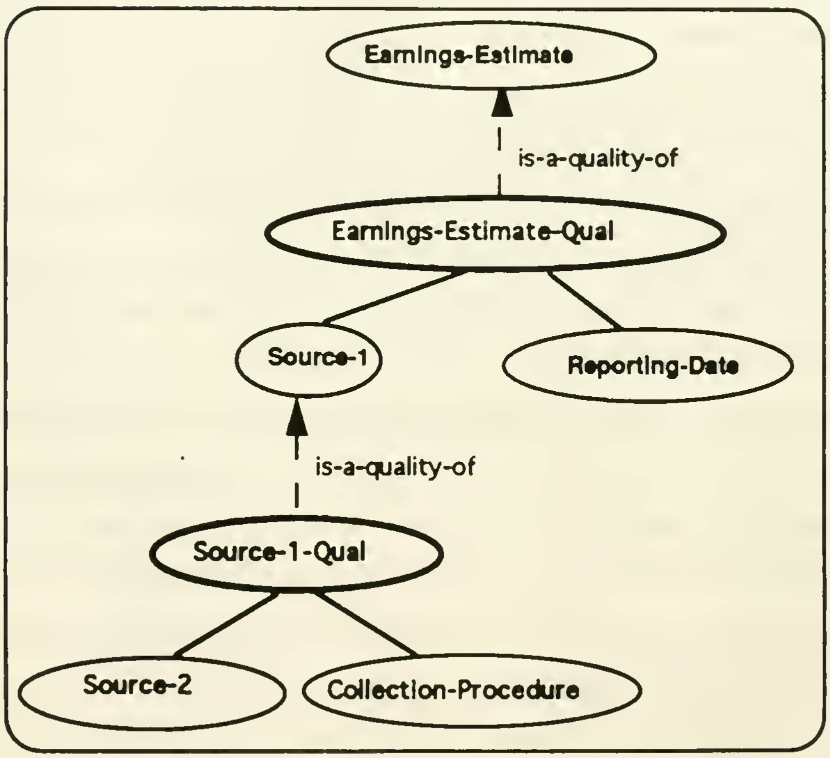

Figure 2 The Quality Data Object Earnings-Estimate

Section 2 presents the quality data object. Section 3 presents a quality data object algebra that allows for the construction of methods which conform with the user's quality requirements. Conclusions and future directions are presented in Section 4.

2. The quality data object

In this section, the quality data object is presented in terms of its structure and behavior. Included in the structure of the quality data object are a definition of the components of a quality data object, the semantics of *is-a-quality-of*, and the quality data object schema. Included in the behavior of the quality data object are a discussion of dimensions of data quality and quality measure methods and messages.

2.1. Structure of the quality data object

Following the object structure defined in the object-oriented paradigm (Banerjee, 1987; Khoshafian & Copeland, 1990; Zdonik & Maier, 1990), we define two object types for the quality data object.

Let I denote the set of system generated identifiers. Let B denote the set of base atomic types such as integer, real, string. Then

- An object is defined as a primitive object provided that its value belongs to B. The value of a primitive object can not be further subdivided. In the context of the quality data object, every datum object is a primitive object.
- An object is defined as a tuple object if its value is of the form $\langle a_1{:}i_1, a_2{:}i_2, ..., a_n{:}i_n \rangle$ where a_i's are distinct attribute names and i_i's are distinct identifiers from I. In the context of the quality data object, every quality description object is a tuple object.

As shown in Figure 1, the quality description object is associated with its datum object through a *is-a-quality-of* link. The composite object resulting from this association is defined as a quality data object which is a unit of manipulation. Thus every quality data object is a composite object. This composite property can be nested in an arbitrary number of levels.

2.1.1. Semantics of the *is-a-quality-of* link

Note that there is no specific mechanism in the object-oriented paradigm to associate the quality description object with the primitive datum object. More specifically, neither the generalization (*is-a*) nor the aggregation (*is-a-part-of*) construct can be used to capture the semantics of the *is-a-quality-of* link. The *is-a* link is used to associate a subclass object with its super class object; and the *is-a-part-of* link is used to associate an object with its assembly object (Banerjee, 1987).

2.1.1.1. Difference between *is-a* and *is-a-quality-of*

The *is-a-quality-of* link is conceptually different from *is-a* because the relation between a datum object and its quality description object is not a super-class vs. subclass relation. It is semantically different from *is-a* because the construct *inheritance* that is associated with *is-a* is not applicable to the *is-a-quality-of* link.

2.1.1.2. Difference between *is-a-part-of* and *is-a-quality-of*

The conceptual difference between *is-a-quality-of* and *is-a-part-of* is that *is-a-part-of* represents the relation between the objects having part and assembly relation; whereas *is-a-quality-of* represents the association between a datum object and its quality description object.

To present the semantic difference between *is-a-quality-of* and *is-a-part-of*, we first discuss the semantics of *is-a-part-of*.

If there is a *is-a-part-of link* from object O_i to object O_j, then O_i is said to have composite reference from O_j. The object O_j is called the parent object of O_i and the object O_i is called the component object of O_j. Based on whether an object has a *is-a-part-of* link with only one object or more than one object, and whether the existence of an object depends on the existence of its parent object, four types of composite references have been formalized (Kim, Bertino, & Garza, 1989): (1) dependent exclusive composite reference, (2) independent exclusive composite reference, (3) dependent shared composite reference, and (4) independent shared composite reference.

The semantic difference between *is-a-part-of* and *is-a-quality-of* comes from the fact that *is-a-quality-of* has only two composite references instead of four in the case of *is-a-part-of*. We refer to them as dependent exclusive quality reference and dependent shared quality reference.

Specifically, let O_d denote a datum object and O_q a quality description object of O_d. Let $Q(O_q)$ denote the set of objects to whom O_q has a *is-a-quality-of* link. Let $del(O_q)$ and $del(O_d)$ denote deletion of O_q and O_d respectively. Then,

- A dependent exclusive quality reference from O_d to O_q means that $Q(O_q) = \{O_d\}$, and $del(O_d)$ implies $del(O_q)$.
- A dependent shared quality reference from O_d to O_q means that $Q(O_q) \supseteq \{O_d\}$. If $Q(O_q) = \{O_d\}$ then $del(O_d)$ implies $del(O_q)$. If $Q(O_q) \supset \{O_d\}$ then $del(O_d)$ implies $\{O_d\}$ is deleted from $Q(O_q)$.

In dependent quality references the quality description object is treated as a weak object and its existence depends on the existence of its corresponding datum object. Dependent exclusive quality references increases storage overhead. Whereas, dependent shared quality references are beneficial from the storage view point but causes problems during deletion and update.

2.1.2. The quality data object schema

Quality data objects are used as building blocks to construct a quality data object schema. For exposition purposes, we first illustrate, in Figure 3, a composite object company in the object-oriented paradigm, which has instance variables Company-Name, CEO-Name, and Earnings-Estimate (each of the instance variables is a primitive object, hence the composite object company).

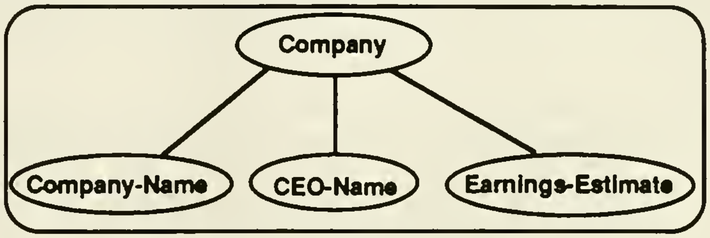

Figure 3: The Object Company

Let us now suppose that out of these three primitive objects, the CEO-Name and Earnings-Estimate are quality sensitive, and are converted into quality data objects as shown in Figure 4 below.

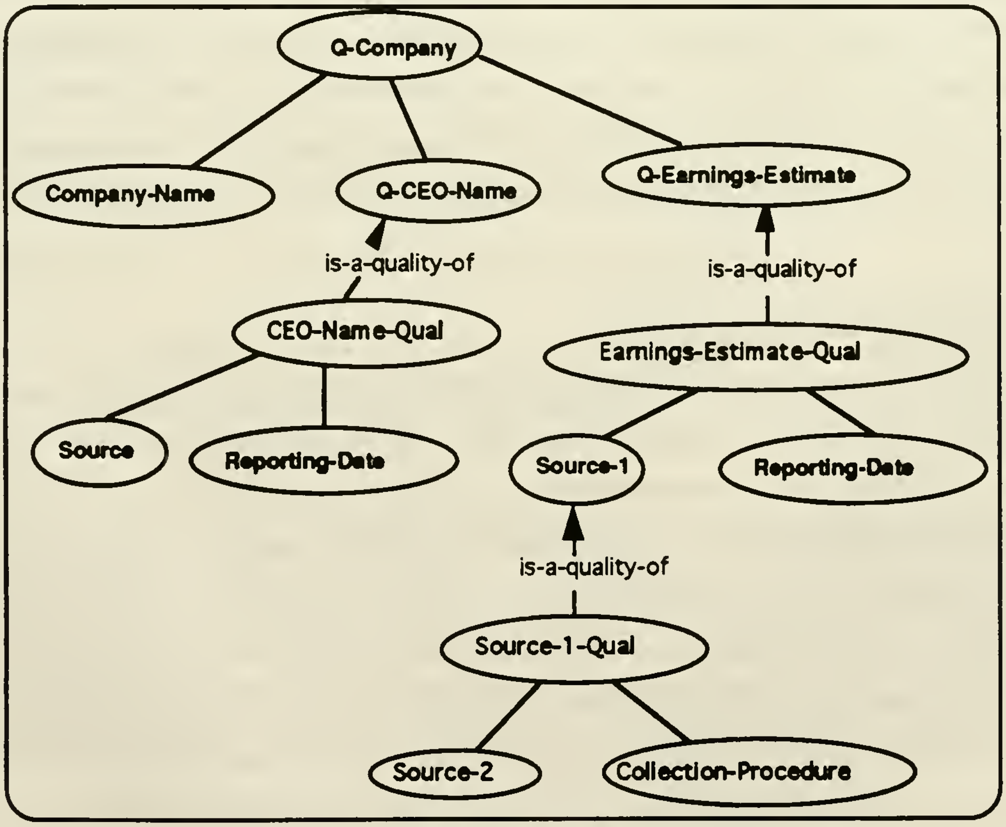

Figure 4: The Quality Data Object Q-Company

The quality data object Q-company is encapsulated as a unit of manipulation. That is, other objects communicate with it through pre-defined methods only. It behaves in the same way as an object in the object-oriented paradigm. In addition, it has the capabilities to measure the quality of data and to retrieve the data that conforms with users' quality requirements, as will be discussed in Section 2.2.

Using quality data objects as basic building blocks, more complex objects can be constructed through other object-oriented constructs such as aggregation (*is-a-part-of*) and generalization (*is-a*). In Figure 5, for example, the Directed Acyclic Graph (DAG) constructed with the quality data objects Q-Company, Q-IT-Department, Q-Finance-Department, and Q-High-Tech-Company forms a quality data object schema.

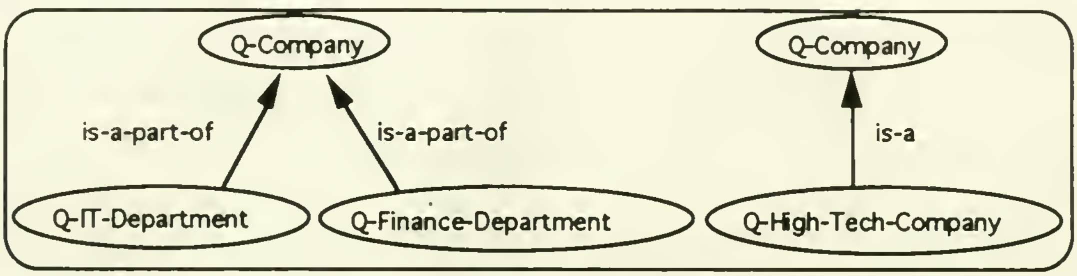

Figure 5: Quality of Object Schema

We have presented the quality data object in terms of its structure. Through the *is-a-quality-of* construct that is unique to the quality data object and the other constructs in the object-oriented paradigm, it is now possible to construct a quality data object schema. The next section presents the behavior of the quality data object that will addresses the issues of how to measure the quality of data.

2.2. Behavior of the quality data object

The multi-dimensional and hierarchical characteristics of data quality were investigated (Wang, Reddy, & Kon, 1992; Wang & Strong, 1992). We illustrate these two characteristics here by considering how a user may make decisions based on certain data retrieved from a database. First the user must be able to get to the data, which means that the data must be accessible (the user has the means and privilege to get the data). Second, the user must be able to interpret the data (the user understands the syntax and semantics of the data). Third, the data must be useful (data can be used as an input to the user's decision making process). Finally, the data must be believable to the user (to the extent that the user can use the data as a decision input). Resulting from this list are the following four dimensions: accessibility, interpretability, usefulness, and believability. In order to be accessible to the user, the data must be available (exists in some form that can be accessed); to be useful, the data must be relevant (fits requirements for making the decision); and to be believable, the user may consider, among other factors, that the data be complete, timely, consistent, credible, and accurate. Timeliness, in turn, can be characterized by currency (when the data item was stored in the database) and volatility (how long the item remains valid) . These multi-dimensional and hierarchical characteristics of data quality provide a conceptual framework for defining the behavior of the quality data object.

In general, the behavior of an object is encapsulated in its methods and messages. In the context of the quality data object, both datum objects and quality description objects will have methods and messages meant for their creation, deletion, and update, just like objects in the object-oriented paradigm.

Each message is described using the following syntax.

Message := ⟨receiver⟩ ⟨message_name⟩([⟨argument⟩])

The ⟨receiver⟩ part is an identifier denoting an object that receives and interpret the message. The ⟨message_name⟩ gives the name of the message which helps the receiving object to associate the message with a particular method. The ⟨argument⟩ part of the message carries data which is required by the method in the receiving object. A message can have zero, one, or more than one arguments, as the brackets " indicate.

Each method is described using the following syntax.

Method_name: ⟨name of the method⟩
Invoked_by: ⟨message[, message]⟩
Method_action: ⟨procedural description of the method⟩

Only those methods and messages related to the data quality aspect are presented in this paper. Below we define key methods that measure data quality.

2.2.1. Currency

The currency dimension is solely a characteristic of storage of the data. We propose to measure currency on a continuous scale from 0 to 1. The state 0 would be assigned to data that are as current as possible, state 1 to the oldest stored data. Let C represent the measure for currency ($0 \le C \le 1$). The value of C is computed dynamically using the creation time of the instance. Creation time is a quality indicator value tagged to every instance. Depending on the message, the currency method can:

- determine the currency of an individual instance,

- determine the average currency of the instances of the class,
- determine the percentage of instances whose currency meets one of the following conditions (referred to as θ) when compared to the total instances of the class available in the database: (1) below or above a user-defined currency level, (2) in between a user-defined currency interval.

A description of the method is given below.

Method_name:	Currency_method
Invoked_by:	⟨receiver⟩ currency(instance_variable)
	⟨receiver⟩ average_currency(instance_variable)
	⟨receiver⟩ θ-currency(instance_variable, θ)
Method_action:	For the message *currency*, the method returns a pairwise value, ⟨instance value, currency⟩, for all the instances satisfying the qualification. For the message *average_currency*, the currency method returns the average currency of all instances of the instance variable. For the message *θ-currency*, the currency method returns the percentage of the currency values of the instances that satisfy the condition θ.

2.2.2. Volatility

The volatility of data is an intrinsic property of the data which is unrelated to its storage time. For example, the fact that George Washington was the first president of the United States remains true no matter how long ago that fact was recorded. On the other hand, yesterday's stock quote may be woefully out of date. We propose to measure volatility on a continuous scale from 0 to 1 where state 0 refers to data that are not volatile at all (they do not change over time) and 1 refers to data that are constantly in flux. The volatility is measured via the coefficient of variation, denoted by V. Let X_i denote a random variable, i = 1, 2, ..., N, then V is computed as follows (Kazmier, 1976):

$$V=\frac{S}{\overline{X}}$$

$$\text{where } S=\sqrt{\frac{\sum_{i=1}^{N} X_i^2 - N\overline{X}^2}{N-1}}$$

$$\text{and } \overline{X}= \frac{1}{N}\sum_{i=1}^{N} X_i$$

A description of the method is given below.

Method_name: Volatility_method

Invoked_by: ⟨receiver⟩ volatility(instance_variable, qualification_for_instances)

Method_action: The system monitors updates to the value of an instance variable that is being modified and simultaneously computes the following three required parameters in order to compute the coefficient of variation: (a) N, the total number of updates, (b) $\overline{X}$, the average of all updated values, and (c) $\sum_{i=1}^{N} X_i^2$, the sum of squares of updated values. These three parameters are stored as quality indicators in the quality description object corresponding to each instance variable. The method returns a pairwise value, ⟨instance value, volatility⟩, for all qualified instances.

2.2.3. Timeliness

Timeliness is defined as a function of currency and volatility of a data value. The most stable situation is to have data for which the currency is 0 (entered very recently) or the volatility of 0 (unchanging) or both. For such data there is no timeliness problem. The worst situation arises when data are old (currency = 1) and highly volatile (volatility = 1). We propose to measure timeliness by combining currency and volatility via their root-mean square: $T=\sqrt{CV}$ where $0 \leq T \leq 1$ with 0 representing the best and 1 the worst case.

A description of the method is given below.

Method_name: Timeliness_method

Invoked_by: ⟨receiver⟩ timeliness(instance_variable, qualification_for_instances)

Method_action: The method returns a pairwise value, ⟨instance value, Timeliness⟩, for for all qualified instances to the message sender.

2.2.4. Accuracy

In general, a user can test the accuracy of the data present in a database with a set of sample data considered to be accurate by the user. For example, a user who wants to check the accuracy of a payroll database can first check whether his salary (known data) is recorded correctly or not. On the basis of this test, the user makes judgment whether to query the database or not. We propose to measure accuracy on a continuous scale from 0 to 1 where state 0 refers to best (all accurate) and 1 the worst (none accurate) case.

A description of the method is given below.

Method_name: Accuracy_method

Invoked_by: ⟨receiver⟩ accuracy(instance_variable, list_of_known_instances)

Method_action: The first argument in the parenthesis gives the name of an instance variable whose accuracy was required. The second argument gives a list of instances that were known to the message sender. This known list of instances is considered as true values. The method computes the percentage, denoted as p, of match between the true values and the recorded values and returns (1-p) to the message sender.

2.2.5. Completeness

Following Ballou and Pazer, we define completeness as all values for a certain variables are recorded (Ballou & Pazer, 1987). We propose to measure completeness on a continuous scale from 0 to 1 where state 0 refers to the best and 1 the worst case. For a given instance variable, the completeness measure 0 implies that it has no null instances in the database, whereas the measure 1 implies all the values recorded for the instance variable are null. Using this, a user can measure the degree of completeness of the database regarding an instance variable.

A description of the method is given below.

Method_name: Completeness_method

Invoked_by: ⟨receiver⟩ completeness(instance_variable)

Method_action: The method measures the percentage, denoted as p, of empty instances of the instance variable when compared to the total instances of the instance variable availabie in the database and returns p to the message sender.

2.2.6. Credibility

The credibility of a datum in a database is computed based on (1) the quality indicator values present in the quality description object of the datum and (2) the set of specifications given by the user.

Let x be an instance. Let q_i be the quality indicator of x and let `J' be the number of quality indictors the user wants to use to compute the credibility of x. Let uv_i be the user's specified value for q_i and let rv_i be the recorded value of the quality indicator q_i for x in the database. Let w_i be the credibility weight assigned to the quality indicator q_i by the user. Let δ_i be a binary variable defined as follows: $\delta_i = 1$ if $uv_i = rv_i$ else $\delta_i = 0$. The credibility of x is computed by the following expression:

$$\sum_{i=1}^{J} w_i * \delta_i.$$

A description of the method is given below.

Method_name: Credibility_method

Invoked_by: (i) ⟨receiver⟩ credibility-a(instance_variable

[, (quality_indicator, quality_indicator_value, credibility_weight)])

(ii) ⟨receiver⟩ credibility-b(instance_variable

[, (quality_indicator, quality_indicator_value)])

(iii) ⟨receiver⟩ credibility-c(instance_variable

[, (quality_indicator, quality_indicator_value, credibility_weight)],

desired_credibility)

Method_action: For the message credibility-a, the method returns values of the instance_variable and their associated credibilities. If weight for each quality_indicator is not specified (message credibility-b) then the method assumes equal weight for each quality indicator specified by the user and returns values of the instance variable and their associated credibilities. For credibility-c, the method returns only those values of the instance variable, whose credibility is more than or equal to the desired_credibility.

We have presented the methods and messages that measure the key dimensions of data quality. They define an important part of the behavior of the quality data object. The other critical behavioral component of the quality data object is the capability to retrieve data that conforms with the user's quality requirements. In the next section, we present an algebra for the quality data object that allows for a systematically construction of retrieval methods for the quality data object.

3. A quality data object algebra

In order to retrieve quality data object instances from a database, it is necessary to identify those quality data object instances that conform with requirements for both the datum portion and the quality description portion. This requires a set of methods to perform the comparisons and an algebra to perform the operations such as selection, projection, and join of quality data objects. Section 3.1 presents quality comparison methods. Section 3.2 presents an algebra that extends the relational algebra to the quality data object domain.

3.1. Quality comparison methods in a quality-description object

In this subsection two different quality comparison methods are discussed in detail and some of the methods which are special cases of these two methods are also discussed, based on equality definitions provided in (Khoshafian & Copeland, 1990). We first define the concepts of *0_deep_equal*, *i_deep_equal*, *M_deep_equal*, and *θ_equal* that underlie these two comparison methods.

Two primitive objects are defined to be *0_deep_equal* if their values matches.

Two tuple objects are defined to be *1_deep_equal* if they have the same set of attributes and if the values they take on the same attribute are *0_deep_equal*. Two tuple objects are defined to be 2_deep_equal if the values they take on same attributes are *1_deep_equal*. Similarly, two tuple objects are defined to be *i_deep_equal* if the values they take on the same attribute are *(i-1)_deep_equal*. Let $o_1 =^i o_2$ denote two tuple objects o_1 and o_2 that is *i_deep_equal*.

In the context of the quality data object, two quality data objects are defined to be *0_deep_equal* if their datum portions are identical. Two quality data objects are *1_deep_equal* if their datum values and the corresponding quality indicator values at the first level are identical. Similarly, two quality data objects are *i_deep_equal* if their datum values and the corresponding quality indicator values up to the i^{th} level are identical. If 'i' is the maximum depth of both o_1 and o_2, and if o_1 and o_2 are *i_deep_equal*, then this relation is defined as *M_deep_equal*, denoted by $o_1 =^M o_2$.

We illustrate the above concepts through Figure 7. In order to do it, we first exemplify the notation used in Figure 7 via Figure 2. Let o_1 be a quality data object in Figure 7. In Figure 2, earnings estimate would correspond to o_1, and the value of earnings estimate would correspond to v_0. Source-1 would correspond to qi_1, and the value of source-1 would correspond to v_1. Source-2 would correspond to qi_{11}, and the value of source-2 would correspond to v_{11}.

In Figure 7, let o_1 and o_2 be two quality data objects. The quality data object o_1 is *0_deep_equal* to the object o_2 because both have the same datum value, v_0. Object o_1 is *1_deep_equal* to o_2 because the values they take on qi_1, qi_2, qi_3 are all the same. However, o_1 is not 2_deep_equal to o_2 because the values they take on qi_{31} are different (v_{31} vs. x_{31}). Since the maximum number of level of o_1 is 2, it follows that o_1 is not ***M_deep_equal*** to o_2.

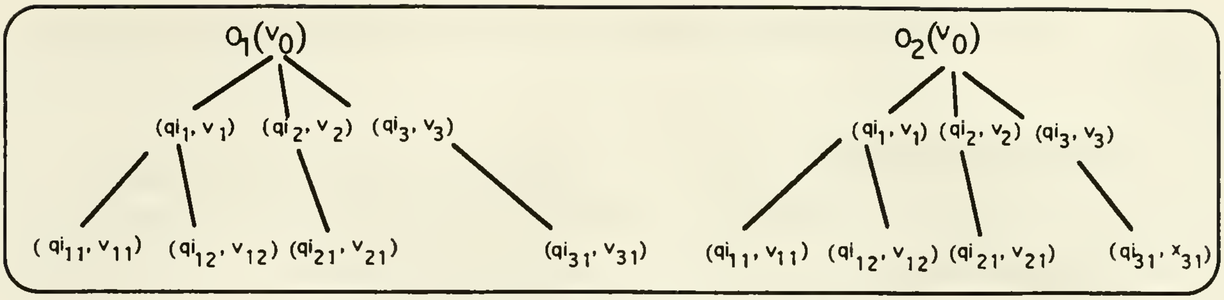

Figure 7: Quality Description objects : o_1 and o_2

We now present the two comparison methods: i*_deep_equal* and θ_equal.

3.1.1. i_deep_equal _method

A description of the method is given below.

Method_name: i_deep_equal _method

Invoked_by: ⟨receiver⟩ i_deep_equal(object$_1$, object$_2$, no_of_levels)

Method_action: This method compares object$_1$ and object$_2$ and then returns True if object$_1$ and object$_2$ are i_deep_equal, where 'i' is the no_of_levels specified by the user, else returns False.

3.1.2. θ_equal _method

Two quality data objects are θ_equal, if the values they take on the attributes in θ are *0_deep_equal*. If two objects o_1 and o_2 have θ_equal then relation is denoted by $o_1 =^{\theta} o_2$.

For example, consider $\theta = \{qi_1, qi_2, qi_{11}, qi_{21}\}$. The quality description objects o_1 and o_2 are θ_equal. Similarly one can define *θ_i_deep_equal_method* and *θ_M_deep_equal_method*. If two objects o_1 and o_2 are θ_i_deep_equal, then relation is denoted by $o_1 =^{\theta\,(i)} o_2$. If two objects o_1 and o_2 have *θ_M_deep_equal*, then relation is denoted by $o1 =^{\theta(M)} o2$. For example, $\theta = \{ qi1 \}$ then o_1 and o_2 have *θ_M_deep_equal*.

A description of the method is given below.

Method_name: θ_equal _method

Invoked_by: ⟨receiver⟩ θ_equal(object$_1$, object$_2$, θ)

Method_action: This method compares object$_1$ and object$_2$ and then returns True if object$_1$ $=^{\theta}$ object$_2$ where θ is the set of quality indicators specified by the user, else returns False.

These two quality comparison methods are used to define the following quality algebraic methods.

3.2. Quality algebraic methods

In this section, we introduce quality algebraic methods to operate on quality data objects.

3.2.1 Selection Method

Selection_method selects only a subset of objects from an object collection such that each object selected must satisfy the selection criterion. Let O be a collection of n objects of type T. Let p and q be first order predicates. This operation creates m (where $m \leq n$) objects of type T from the members of collection O, which satisfy the predicates p and q. The predicate p is a constraint on the datum object and the predicate q is a constraint on the quality description object. The selection_method symbolically denoted as $\sigma^q(O,p,q)$, is defined as follows:

$$\sigma^q (O, p,q) = \{o \mid (o \in O) \wedge p(o) \wedge q(o)\}$$

A description of the method is given below.

Method_name: Selection_Method

Invoked_by: ⟨receiver⟩ selection(object_class, data_constraint, quality_constraint)

Method_action: This method checks each instance of the object_class to see whether they satisfy data_constraint and the quality_constraint, and returns all object instances of the object_class which satisfy both of these constraints.

3.2.2 Union Method

In union_method, the two operand quality data object collections must be of the same type. Let O_1 be the collection of n objects of type T and O_2 be the collection of m objects of type T. The result of this method is a collection of p objects (where $n \leq p \leq n+m$) of type T. This method selects all instances from the collection O_1 and selects only those instances from the collection O_2 which are not duplicates when compared to the instances of O_1. The logic of the union_method, which is symbolically denoted as $\cup^q (O_1, O_2, \theta)$ is defined as follows

$$\cup^q (O_1 , O_2 , \theta) = \{o \mid \forall o \in O_1 \} \cup \{ o \mid \forall o_2 \in O_2 \exists o_1 \in O_1 \wedge (o =^M o_2) \wedge \neg \{(o_1 =^0 o_2) \wedge (o_1 =^\theta o_2)\} \}$$

In the above expression, " $\neg (o_1 =^0 o_2) \wedge (o_1 =^\theta o_2)\}$ " is meant to eliminate duplicates. Objects o_1 and o_2 are considered duplicates provided that their datum portions are the same and they are *0_deep_equal* with respect to all quality indicators in θ. Note that the above definition for the union is commutative from the view point of the user who defined θ. In general it is not commutative because, $o_1 =^{\theta} o_2$ does not mean $(o_1 =^{M} o_2)$.

A description of the method is given below.

Method_name: Union_method

Invoked_by: ⟨receiver⟩ union(O_1, O_2,θ)

Method_action: Let *result1* be the set of all instances in the object collection O_1. Let *result2* be the subset (need not be strict subset) of instances of O_2 such that any instance from *result2* is not 0_deep_equal and not θ_equl to any instance in *result1*. Let *result* =*result1* ∪ *result2*. This method returns the set *result*.

3.2.3 Difference Method

In difference_method, the two operand object collections must be of the same type. Let O_1 be a collection of n objects of type T and O_2 be a collection of m objects of type T. The result of the difference method is a collection of p objects (where $p \leq n$) of type T. The result consists of objects only from O_1 which are not *0_deep_equal* to objects in O_2 with respect to all quality indicator specified in θ. The logic of the difference_method, denoted as $-^{q}(O_1, O_2, \theta)$ is defined as follows

$$-^{q}(O_1, O_2, \theta) = \{ o \mid \forall o_1 \in O_1 \; \exists o_2 \in O_2, (o =^{M} o_1) \wedge \neg \{(o_1 =^{0} o_2) \wedge (o_1 =^{\theta} o_2)\}\}$$

A description of the method is given below.

Method_name: Difference_method

Invoked_by: ⟨receiver⟩ difference(O_1, O_2,θ)

Method_action: Let *result* be the set of all the instances of O_1 except those that are 0_deep_equal and θ_equal to any instance of O_2. This method returns the set *result*.

3.2.4 Projection Method

Let O be an object collection of m objects of type T. Projection_method generates p (where $p \leq m$) objects of type T′ from the object collection O. Let o be an object in the collection O. The function f returns an object o′ of type T′ from the object o. The projection method also eliminates duplicate objects from the result. The logic of the projection_method, which is symbolically denoted as $\Pi^{q}(O, f{:}T', \theta)$ is defined as follows

$$\Pi^{q}(O, f{:}T', \theta) = \{\{\Pi(O, f{:}T')\} - \{o_1\} \mid o_1, o_2 \in \Pi(O, f{:}T'), \{(o_1 =^{0} o_2) \wedge (o_1 =^{\theta} o_2)\}\}$$

where $\Pi(O, f{:}T') = \{f(o) \mid o \in O\}$

A description of the method is given below.

Method_name: Projection_method

Invoked_by: ⟨receiver⟩ projection(O, θ)

Method_action: Let O′ be the object type whose instances variables are a subset of the instance variables of O. Let f be a function which takes an instance of O and instantiates O′. Let *result1* be the set of instances of O′. Let *result* ⊆ *result1* be the set of instances generated by eliminating duplicates from the set *result1*. The method returns the set *result*.

3.2.5 Cartesian Product Method

Let O_1 be an object collection of n objects of type T_1, and let O_2 be an object collection of m objects of type T_2. Let o_1 be an object in the collection O_1 and let o_2 be an object in the collection O_2. The method constructs a new object $o_1 \oplus o_2$ of type T3, from o_1 and o_2. Objects of type T_3 consists of instance variables from both T_1 and T_2. The logic of the cartesian_product_method, denoted as $\Pi^q(O, f{:}T', \theta)$ is defined as follows

$$X^q(O_1, O_2) = \{o \mid \forall o_1 \in O_1 \ \ \forall o_2 \in O_2, o = o_1 \oplus o_2\}$$

A description of the method is given below.

Method_name: Cartesian_product_method

Invoked_by: ⟨receiver⟩ cartesian_product(O_1, O_2)

Method_action: Let O_3 be a new object type which will have all the instance variables of O_1 and of O_2. Let f be the function which take instances of O_1 and instances of O_2 and with these instances, the function f instantiates the object type O_3. This method returns the set of instances of O_3.

Other algebraic methods such as Intersection_method and Join_method can be defined using the above defined five algebraic methods.

4. Concluding remarks

In this paper, we have investigated how to associate data with quality information that can help users make judgments of the quality of data for the specific application at hand. Our research question was how to structure and manage data in such a way that users could be equipped with the capabilities to measure the quality of data they need and to retrieve the data that conforms with their quality requirements.

Toward this goal, we have proposed the concept of quality data object in which each datum object is associated with appropriate data and procedures used to indicate the quality of the datum object. Specifically, the *is-a-quality-of* link is proposed to associate a datum object with its corresponding quality description object. The composite object constructed from a datum object and its associated quality description object is called a quality data object. It provides methods which can access object instances which matches users' quality requirements. It also provides a set of quality measure methods that compute quality dimension values including currency, volatility, timeliness, accuracy, consistency, and completeness. In addition, we have developed a quality data object algebra that includes quality comparison methods and an algebra that extends the relational algebra to the quality data object domain. It allows for a systematic construction of retrieval methods for quality data objects.

The concept of quality data object presented in the paper is a first step toward the design and manufacture of data products. We envision that the quality data object proposed in this paper can be used as basic building blocks for the design, manufacture, and delivery of quality data products. It will enable users to measure the quality of data products according to their chosen criteria; it will also enable users to purchase data products based on their quality requirements. In this manner, we hope that the concepts of quality data objects and quality data products will help improve data quality and data reusability.

5. Appendix A

Many concepts in the object-oriented paradigm can be applied to support the quality data object. They are fundamental in our decision to model the quality data object via the object-oriented approach. In this Appendix we discuss how constructs in the object-oriented paradigm such as inheritance, method, polymorphism, active value, and message can be exploited to support the quality data object.

We present features of the object-oriented paradigm and relate them to the quality data object.

Modeling Paradigm In the object-oriented paradigm, all conceptual entities are modeled as objects (Kim, 1989; Kim, 1990). This paradigm is particularly interesting to us because both data and its quality can be represented as objects, as Figures 1-2 illustrate. It eliminates the dichotomy of representation schemes for data and its quality. In Figure 2, for example, the datum object Earnings-Estimate is modeled as an object and its quality description attributes such as Source-1 and Reporting-Date are also modeled as objects.

Inheritance Objects in an object hierarchy can inherit both the data and methods from their parent objects in the object hierarchy (Banerjee, 1987; Snyder, 1986; Zdonik & Maier, 1990). In the context of the quality data object, whenever a quality data object is inherited by its child object, the quality information is automatically inherited. Therefore, both quality indicators and quality procedures can be reused just like data and methods in the object-oriented paradigm.

Method The behavior of an object in the object-oriented paradigm is encapsulated in methods (Banerjee, 1987; Zdonik & Maier, 1990). A method consists of code that manipulates and returns the state of an object. In the context of the quality data object, mechanisms used to determine data quality dimension values are procedure-oriented, and are difficult to express declaratively. Therefore, the procedural capability in the object-oriented paradigm can be used effectively to define quality procedures in a quality data object. For example, timeliness of a quality data object is procedure-oriented and can be encapsulated as a method. As another example, since objects are instantiated, deleted, and modified by the methods of the object, the corresponding quality integrity constraints (Wang, Reddy, & Kon, 1992) can be embedded in the definition of these methods.

Polymorphism In the object-oriented paradigm, the same method name can be used in different objects to define different procedures, and the same method can take different types or different number of arguments (Zdonik & Maier, 1990). This feature is important in the context of the quality data object because data quality measure methods can be defined differently in different objects with the same name. Moreover, the evaluation of a procedure depends on the type and number of arguments passed to the procedure which, in turn, depend on the quality requirements of a user. For example, the method believability can be invoked with different sets of arguments: One user may believe the Earnings-Estimate if the immediate source (e.g. the Wall Street Journal) is credible whereas another user may consider additional quality indicators such as source of source (e.g., the Wall Street Journal quoted Zacks Investment Research which is considered very credible by the investment community) as important in determining the believability. Using polymorphism, both of the users can use the same method but with different sets of arguments.

Active values In the object-oriented paradigm, the values of active instance variables are computed at run time based on values of other instance variables (Zdonik & Maier, 1990). This feature is useful in computing data quality dimension values dynamically. Since data quality, in a sense, lies in the eyes of the beholder (Wang, Kon, & Madnick, 1993), some quality dimensions of a quality data object need to be computed dynamically based on (1) user requirements and (2) data and procedures encapsulated in the quality data object. For example, timeliness of a quality data object can not be stored as a value. It must be computed dynamically upon demand, as discussed in Section 2.

Messages In the object-oriented paradigm, objects can communicate with one another through messages (Maier & Stein, 1987). Messages, together with any arguments that may be passed with the messages, constitute the public interface of an object. This feature is handy in the context of quality data object because the extra complexity introduced in a quality data object can be encapsulated by the interface of an object which is nothing but a collection of messages.

6. References

[1] Ballou, D. P. & Pazer, H. L. (1987). Cost/Quality Tradeoffs for Control Procedures in Information Systems. *International Journal of Management Science*, 15(6), pp. 509-521.

[2] Banerjee, J., et al,. (1987). Data Model Issues for Object-Oriented Applications. *ACM Transactions on Office Information Systems*, 5(1).

[3] Bernstein, P. A. & Goodman, N. (1981). Concurrency Control in Distributed Database Systems. *Computing Surveys*, 13(2), pp. 185-221.

[4] Bernstein, P. A., et al. (1981). Query Processing in a System for Distributed Databases (SDD-1). *ACM Transactions on Database Systems*, 6(4), pp. 602-623.

[5] Chen, P. P. (1976). The Entity-Relationship Model - Toward a Unified View of Data. *ACM/TODS*, 1, pp. 166-193.

[6] Chen, P. P. (1984). *An Algebra for a Directional Binary Entity-Relationship Model.* Los Angeles, CA. 1984. pp. 37-40.

[7] Chen, P. P. (1991). *Entity-Relationsip Approach to Database Design* . Wellesley, MA: Q.E.D. Information Sciences.

[8] Chen, P. P. & Li, L. (1987). The Lattice Concept in Entity Set. In S. Spaccapaietra (Ed.), *Entity-Relationship Approach* North-Holland.

[9] Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, 13(6), pp. 377-387.

[10] Codd, E. F. (1979). Extending the relational database model to capture more meaning. *ACM Transactions on Database Systems*, 4(4), pp. 397-434.

[11] Codd, E. F. (1982). Relational database: A practical foundation for productivity, the 1981 ACM Turing Award Lecture. *Communications of the ACM*, 25(2), pp. 109-117.

[12] Codd, E. F. (1986). *An evaluation scheme for database management systems that are claimed to be relational*. The Second International Conference on Data Engineering, Los Angeles, CA. 1986. pp. 720-729.

[13] Date, C. J. (1981). *Referential Integrity*. The Proceedings of the 7th International Conference on Very Large Data bases, Cannes, France. 1981.

[14] Date, C. J. (1985). *An Introduction to Database Systems* . Reading, MA: Addison-Wesley.

[15] Date, C. J. (1990). *An Introduction to Database Systems* (5th ed.). Reading, MA: Addison-Wesley.

[16] Denning, D. E. & Denning, P. J. (1979). Data Security. *ACM Comp. Surv.*, 11(3).

[17] Fernandez, E. B., Summers, R. C., & Wood, C. (1981). *Database Security and Integrity* . Readings, MA: Addison-Wesley.

[18] Hoffman, L. J. (1977). *Modern Methods for Computer Security and Privacy* . Englewood Cliffs, NJ: Prentice Hall.

[19] Hsiao, D. K., Kerr, D. S., & Madnick, S. M. (1978). *Privacy and Security of Data Communications and Data bases*. Proceedings of the 4th International Conference on Very Large Data Bases, 1978.

[20] Huh, Y. U., et al. (1990). Data Quality. *Information and Software Technology*, 32(8), pp. 559-565.

[21] Johnson, J. R., Leitch, R. A., & Neter, J. (1981). Characteristics of Errors in Accounts Receivable and Inventory Audits. *Accounting Review*, 56(April), pp. 270-293.

[22] Kazmier, L. J. (1976). *Business Statistics* . New York, N.Y.: McGraw Hill Company.

[23] Khoshafian, S. N. & Copeland, G. P. (1990). Object Identity. In S. B. Zdonik& D. Maier (Ed.), (pp. 37-46). San Mateo, CA: Morgan Kaufmann.

[24] Kim, W., Lochovsky, Frederick H. (1989). *Object-Oriented Concepts, Databases, and Applications* (1 ed.). New York: Addison-Wesley.

[25] Kim, W. (1990). Object-Oriented Databases: Definition and Research Directions. *IEEE Transactions on Knowledge and Data Engineering*, 2(3).

[26] Kim, W., Bertino, E., & Garza, J. F. (1989). *Proceedings of the 1989 ACM SIGMOD Internaltional Conference on the Management of Data.* ACM SIGMOD, Portland, Oregan. 1989. pp. 337-347.

[27] Korth, H. & Silberschatz, A. (1986). *Database System Concepts* . New York: McGraw-Hill Book Company.

[28] Laudon, K. C. (1986). Data Quality and Due Process in Large Interorganizational Record Systems. *Communications of the ACM*, 29(1), pp. 4-11.

[29] Liepins, G. E. & Uppuluri, V. R. R. (1990). *Data Quality Control: Theory and Pragmatics* (pp. 360). New York: Marcel Dekker, Inc.

[30] Liepins, O. E. (1989). Sound Data Are a Sound Investment. *Quality Programs*, (September), pp. 61-63.

[31] Maier, D. & Stein, J. (1987). *Development and Implentation of an Object-Oriented DBMS* . Cambridge, MA: MIT Press.

[32] Martin, J. (1973). *Security, Accuracy, and Privacy in Computer Systems* . Englewood Cliffs, NJ: Prentice Hall.

[33] Maxwell, B. S. (1989). Beyond "Data Validity": Improving the Quality of HRIS Data. *Personnel*, , pp. 48-58.

[34] McCarthy, J. L. (1982). *Metadata Management for Large Statistical Databases*. Mexico City, Mexico. 1982. pp. 234-243.

[35] McCarthy, J. L. (1984). *Scientific Information = Data + Meta-data*. U.S. Naval Postgraduate School, Monterey, CA. 1984.

[36] McCarthy, J. L. (1988). *The Automated Data Thesaurus: A New Tool for Scientific Information*. Proceedings of the 11th International Codata Conference, Karlsruhe, Germany. 1988.

[37] Qian, X. & Wiederhold, G. (1986). *Knowledge-based integrity constraint validation*. The Twelfth International Conference on Very Large Data Bases, 1986. pp. 3-12.

[38] Redman, T. C. (1992). *Data Quality: Management and Technology* . New York, NY: Bantam Books.

[39] Sciore, E. (1991). Using Annotations to Support Multiple Kinds of Versioning in an Object-Oriented Database System. *ACM Transactions on Database Systems*, 16(No. 3, September 1991), pp. 417-438.

[40] Siegel, M. & Madnick, S. E. (1991). *A metadata approach to resolving semantic conflicts*. Barcelona, Spain. 1991.

[41] Snyder, A. (1986). *Encapsulation and Inheritance in Object-Oriented Programming Languages*. OOPSLA-86, 1986. pp. 38-45.

[42] Ullman, J. D. (1982). *Principles of Database Systems* . Rockville, Maryland, USA: Computer Science Press.

[43] Wang, R. Y. & Kon, H. B. (1992). Towards Total Data Quality Management (TDQM). In R. Y. Wang (Ed.), *Information Technology in Action: Trends and Perspectives* Englewood Cliffs, NJ: Prentice Hall.

[44] Wang, R. Y., Kon, H. B., & Madnick, S. E. (1993). *Data Quality Requirements Analysis and Modeling in Data Engineering.* in the Proceedings of the 9th International Conference on Data Engingeering, Vienna. 1993.

[45] Wang, R. Y., Reddy, M. P., & Kon, H. B. (1992). Toward Quality Data: An Attribute-based Approach. *to appear in the Journal of Decision Support Systems (DSS), Special Issue on Information Technologies and Systems,.*

[46] Wang, Y. R. & Madnick, S. E. (1990). *A Polygen Model for Heterogeneous Database Systems: The Source Tagging Perspective.* Brisbane, Australia. 1990. pp. 519-538.

[47] Wang, Y. R. & Strong, D. (1992). Dimensions of Data Quality: Beyond Accuracy. *Submitted for Publication.*

[48] Zarkovich. (1966). *Quality of Statistical Data* . Rome: Food and Agriculture Organization of the United Nations.

[49] Zdonik, S. B. & Maier, D. (1990). *Readings in Object OIriented Database Systems* (pp. 629). California: Morgan Kaufmann Publishers, Inc.