A FORMAL DEFINITION OF DATA QUALITY PROBLEMS

(Completed paper)

Paulo Oliveira
DI/gEPL – Languages Specification and Processing Group
University of Minho (Portugal), and
GECAD/ISEP-IPP – Knowledge Engineering and Decision Support Group
Institute of Engineering – Polytechnic of Porto (Portugal)
pjo@isep.ipp.pt

Fátima Rodrigues
GECAD/ISEP-IPP - Knowledge Engineering and Decision Support Group
Institute of Engineering – Polytechnic of Porto (Portugal)
mfc@isep.ipp.pt

Pedro Henriques
DI/gEPL – Languages Specification and Processing Group
University of Minho (Portugal)
prh@di.uminho.pt

Abstract: The exploration of data to extract information or knowledge to support decision making is a critical success factor for an organization in today's society. However, several problems can affect data quality. These problems have a negative effect in the results extracted from data, affecting their usefulness and correctness. In this context, it is quite important to know and understand the data problems. This paper presents a taxonomy of data quality problems, organizing them by granularity levels of occurrence. A formal definition is presented for each problem included. The taxonomy provides rigorous definitions, which are information-richer than the textual definitions used in previous works. These definitions are useful to the development of a data quality tool that automatically detects the identified problems.

Key Words: Data Quality Problems, Formal Definition, Taxonomy

1. INTRODUCTION

Nowadays, public and private organizations understand the value of data. Data is a key asset to improve efficiency in today's dynamic and competitive business environment. However, as organizations begin to create integrated data warehouses for decision support, the resulting Data Quality (DQ) problems become painfully clear [12]. A study by the Meta Group revealed that 41% of the data warehouse projects fail, mainly due to insufficient DQ, leading to wrong decisions [8]. The quality of the input data strongly influences the quality of the results [15] ("garbage in, garbage out" principle).

The concept of DQ is vast, comprising different definitions and interpretations. DQ is essentially studied in two research communities: databases and management. The first one studies DQ from a technical point of view (*e.g.*, [4]), while the second one is also concerned with other aspects or dimensions (*e.g.*, accessibility, believability, relevancy, interpretability, objectivity) involved in DQ (*e.g.*, [13, 17]). In the context of this paper we follow the databases perspective, *i.e.*, DQ means just the quality of the data values or instances.

DQ problems are also labeled of errors, anomalies or even *dirtiness* and enclose, among others, missing attribute values, incorrect attribute values, or different representations of the same data. It is not uncommon for operational databases to have 60% to 90% of bad data [3]. These problems are an obstacle to effective data usage and, as already said, negatively affect the results and conclusions obtained. Therefore, before using an analysis-oriented tool, data must be examined to check whether the required quality is assured. If not, DQ must be improved by removing or repairing any problems that may exist [10].

DQ problems concerns arise in three different contexts [4]: (*i*) when one wants to correct anomalies in a single data source, as files and databases (*e.g.*, duplicate elimination in a file); (*ii*) when poorly structured or unstructured data is migrated into structured data; or (*iii*) when one wants to integrate data coming from multiple sources into a single new data source (*e.g.*, data warehouse construction). In the last situation, the problems are even more critical because distinct data sources frequently contain redundant data under different representations. The representations must be consolidated and the duplicates removed to provide an accurate and consistent access to data.

The paper reports the new developments of our work, initially presented in [11]. The main contributions provided here are: (*i*) a formal definition for each DQ problem and (*ii*) a taxonomy of DQ problems that organizes them by granularity levels of occurrence.

In [7, 9, 14] a comprehensive list of problems that affect DQ is presented. However, the problems are only described through a textual definition. It is commonly accepted that natural language tends to be ambiguous by nature. Therefore, doubts about the true meaning of some DQ problems arise, which means that they need to be further clarified. Using a formal definition is a suitable approach to specify each DQ problem in a rigorous way.

Besides rigorous, this kind of definition is also useful because has more extra information than a textual definition. The definition makes explicit: (*i*) that it concerns just with a given data type (*e.g.* string data type); (*ii*) the metadata knowledge needed to detect a problem (*e.g.* the attribute domain); (*iii*) the mathematical expression that specifies the DQ problem, which can be computationally translated to automate its detection (just for illustration purposes we show the definition of *domain violation* later presented: $\exists\ t \in r : v(t,a) \notin Dom(a)$); and (*iv*) eventually a required function that allows to detect the DQ problem (*e.g.* to detect a misspelling error, a spell checker function must be available). A framework for DQ problems is our first step towards the development of an automated tool for detecting the problems. With this tool we intend to complement the capabilities of today's data profiling tools.

We argue that a taxonomy of DQ problems is important because: (*i*) it is useful to understand how far a given DQ tool is able to go in detecting and correcting DQ problems, *i.e.*, it allows to measure the coverage of a DQ tool; (*ii*) guides the research efforts, emphasizing the DQ problems that deserve further attention, *i.e.*, if a DQ problem has no detection or correction support, this means that research attention should be given to it.

The paper is organized as follows. Section 2 presents in detail our taxonomy, organized by granularity levels of DQ problems. Section 3 compares our taxonomy to related work. Finally, in Section 4, conclusions and some future work directions are described.

2. DATA QUALITY PROBLEMS

Figure 1 presents the well known typical model of data organization: (*i*) data is stored in multiple data sources; (*ii*) a data source is composed of several relations and relationships are established among them; (*iii*) a single relation is made up of several tuples; and (*iv*) a tuple is composed by a predefined number of attributes. This model results in a hierarchy of four levels of data granularity: multiple data sources; multiple relations; single relation; and attribute/tuple.

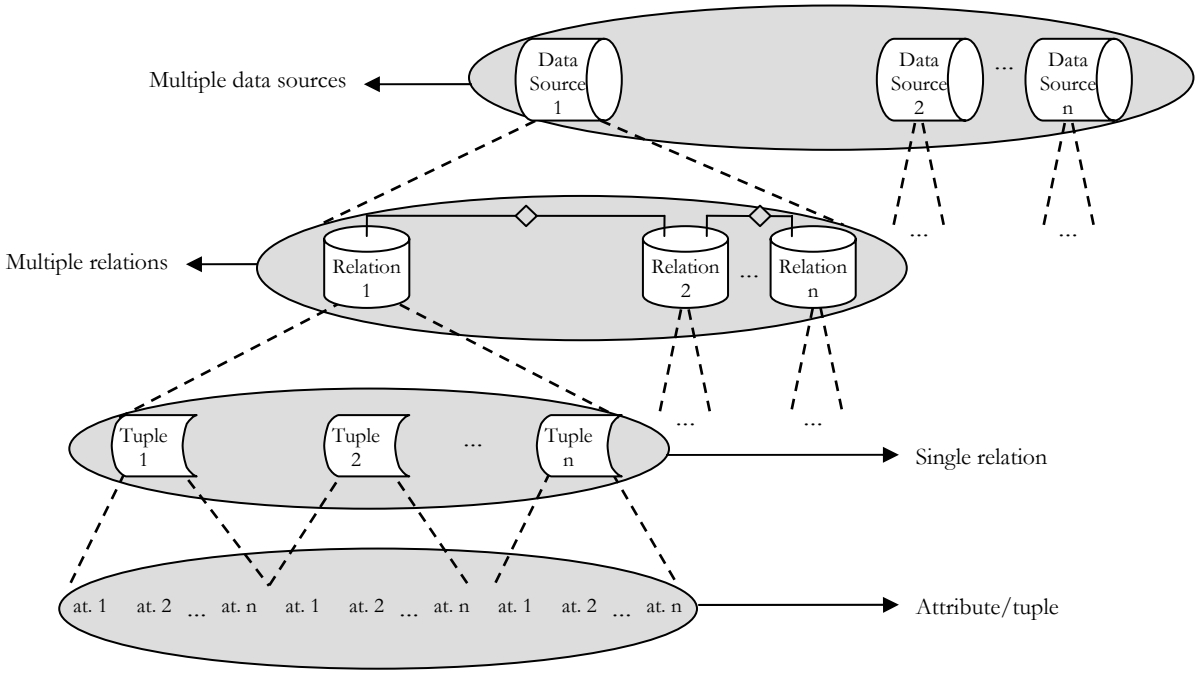

Figure 1: Typical model of data organization

We have identified the DQ problems and created the taxonomy based on this model. Using real-world data from the retail sector, we thoroughly analyzed each granularity level, from the lowest (attribute/ tuple) to the highest (multiple data sources), to detect specific DQ problems. The purpose was to tackle first the most specific and easier to detect DQ problems and leave to the end the most generic and difficult ones. The analysis was based on the fundamental elements of this model of data organization (*e.g.*, data types, relationships, data representation structure). The systematic approach used supports our conviction that the taxonomy is complete.

The DQ problems identified are presented through a rigorous definition and properly illustrated with an example. Our taxonomy covers the problems that affect data represented in a tabular format, *i.e.*, at least in the first normal form. The types of data considered are: numeric, date/time, enumerated, and string. Multimedia data was excluded, since it requires a special kind of manipulation.

2.1 Preliminaries

We start by introducing the notation used throughout the paper, following [1]. A *relation schema* consists of a name R (the relation name), along with a list $A = a_1,a_2,...,a_n$ of distinct attribute names, represented by $R(a_1,a_2,...,a_n)$ or simply by $R(A)$. The integer n represents the relation schema *degree*. A *data source DS* is composed by a set of m relation schemas $R_1(A_1)$, $R_2(A_2),...,R_m(A_m)$. A *domain d* is a set of atomic values. Given a set D of domains, a set A of attribute names, we assume there is a function Dom: $A \rightarrow D$ that associates attributes with their domains. The function Dom is applied to the set of attribute names, which is represented by $Dom(A) = Dom(a_1,a_2,...,a_n) = Dom(a_1) \times Dom(a_2) \times ...\times Dom(a_n)$. A *relation instance* (or

relation, for short) is a finite set $r \subseteq Dom(a_1) \times Dom(a_2) \times ... \times Dom(a_n)$ and is represented by $r(a_1,a_2,...,a_n)$ or simply by $r(A)$. Each element of r is called a *tuple*, and is represented by t. A tuple t can be viewed as a function that associates a value of $Dom(a_1)$ with a_1, a value of $Dom(a_2)$ with a_2,... and a value of $Dom(a_n)$ with a_n. The value of attribute a in tuple t is represented by $v(t,a)$. The values in tuple t of attributes $a_1,a_2,...,a_n$ *i.e.*, $v(t,a_1),v(t,a_2),...,v(t,a_n)$ are denoted by $v(t,A)$. By analogy, the values attribute a take for all tuples are represented by $v(T,a)$. The application of a function f over a value v is represented by $f(v)$ while over a set of values V is represented by $f(V)$. If, for some reason, a value transformation cannot be done, the function f acts as the identity function, *i.e.*, $f(v) = v$ or $f(V) = V$. The data type of attribute a is denoted by $type(a)$.

2.2 DQ Problems at the Level of Attribute/Tuple

This level is divided into three groups of DQ problems that were encountered by analyzing the value(s) of: (*i*) a single attribute of a single tuple; (*ii*) a single attribute in multiple tuples (a column); and (*iii*) multiple attributes of a single tuple (a row).

2.2.1 Single Attribute of a Single Tuple

The following DQ problems were detected by analyzing the values of single attributes (with different data types) in single tuples (as presented in Figure 2).

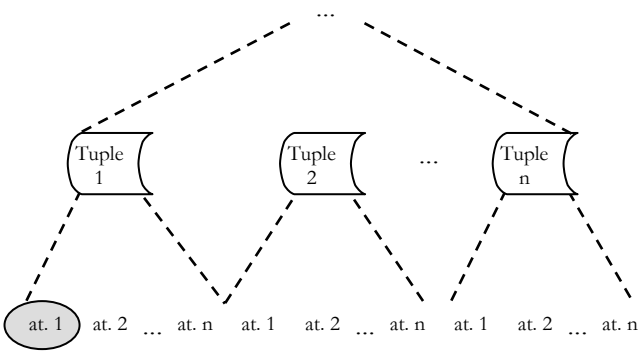

Figure 2: A single attribute of a single tuple

- **Missing value**

  Definition: Let S be a set of attribute names, defined as: $S = \{a \mid a \in R(A) \wedge a$ is a mandatory attribute$\}$, *i.e.*, $S \subseteq R(A)$. There is a *missing value* in attribute $a \in S$ if and only if (*iff*): $\exists\ t \in r : v(t,a) = null$.

  Example: Absence of value in the mandatory attribute *Name* of a customer.

  Note: The absence of value in an optional attribute is not considered by us as a DQ problem.

- **Syntax violation**

  Definition: Let $G(a)$ be the syntax of attribute a, given by a grammar or a regular expression. Let $\mathbb{L}(G(a))$ be the language generated by the grammar or regular expression. There is a *syntax violation* in attribute $a \in R(A)$ *iff*: $\exists\ t \in r$: $v(t,a) \notin \mathbb{L}(G(a))$.

  Example: The attribute *Order_Date* contains the value *13/12/2004*, instead of *2004/12/13*.

- **Incorrect value**

  Definition: Let $u(t,a)$ be the correct and updated value that the attribute a of tuple t was supposed to have. There is an *incorrect value* in attribute $a \in R(A)$ *iff*: $\exists\ t \in r : v(t,a) \in Dom(a) \wedge v(t,a) \neq u(t,a)$.

Example: The attribute *Creation_Date* contains the value *23/09/2003*, instead of *23/09/2004*.

- **Domain violation**

  Definition: There is a *domain violation* in attribute $a \in R(A)$ *iff*: $\exists\, t \in r : v(t,a) \notin Dom(a)$.

  Example: In a given order, the attribute *Ordered_Quantity* contains a negative value.

- **Violation of business domain constraint**

  Definition: Let *check* be a function that receives an attribute value, checks whether it respects a given constraint, and returns a boolean value. There is a *violation of business domain constraint* in attribute $a \in R(A)$ *iff*: $\exists\, t \in r : check(v(t,a)) = false$.

  Example: The attribute *Name* of a customer must have, at least, two words; however, in a certain tuple this constraint is not respected.

A *domain violation* in an attribute whose data type is string, may be further detailed as presented next.

Let S be a set of attribute names, defined as: $S = \{a \mid a \in R(A) \wedge type(a) = \text{string}\}$, *i.e.*, $S \subseteq R(A)$. This set is used in the following definitions.

- **Invalid substring**

  Definition: Let $v'(t,a)$ be a substring of $v(t,a)$. There is an *invalid substring* in attribute $a \in S$ *iff*: $\exists\, t \in r : v(t,a) \notin Dom(a) \wedge v'(t,a) \in Dom(a)$.

  Example: The attribute *Customer_Name* also stores the academic degree (e.g., Dr. John Taylor).

- **Misspelling error**

  Definition: Let *spell* be a spelling checker function that receives a misspelled word, looks-up for the correct word based on a language dictionary, and returns it. There is a *misspelled error* in attribute $a \in S$ *iff*: $\exists\, t \in r : v(t,a) \notin Dom(a) \wedge spell(v(t,a)) \in Dom(a)$.

  Example: The attribute *Address_Place* contains the value *Sant Louis*, instead of *Saint Louis*.

- **Imprecise value**

  Definition: Let *translate* be a function that receives an abbreviation or an acronym, looks-up for its meaning (in full words) in a dictionary (lookup table), and returns it. There is an *imprecise value* in attribute $a \in S$ *iff*: $\exists\, t \in r : v(t,a) \notin Dom(a) \wedge translate(v(t,a)) \in Dom(a)$.

  Example: The value *Ant.* in attribute *Customer_Contact* may represent *Anthony*, *Antonia*, etc.

2.2.2 Single Attribute in Multiple Tuples

The following DQ problems were identified by analyzing the values of a single attribute in multiple tuples, as illustrated in Figure 3.

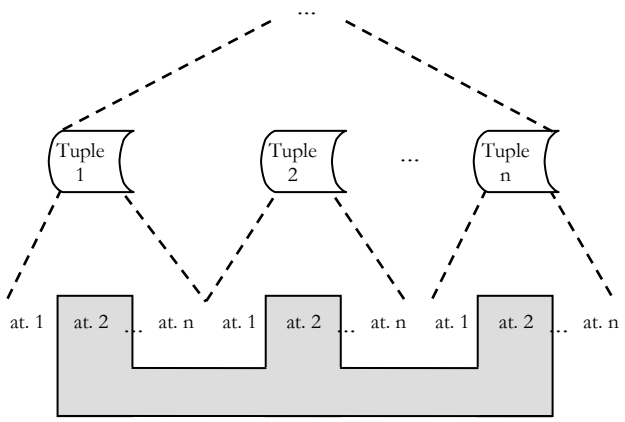

Figure 3: A single attribute in multiple tuples

- **Unique value violation**

  Definition: Let S be a set of attribute names, defined as: $S = \{a \mid a \in R(A) \wedge a$ is a unique value attribute$\}$, *i.e.*, $S \subseteq R(A)$. There is a *unique value violation* in attribute $a \in S$ *iff*: $\exists\ t_1, t_2 \in r : v(t_1,a) = v(t_2,a) \wedge t_1 \neq t_2$.

  Example: Two different customers have the same taxpayer identification number.

- **Existence of synonyms**

  Definition: Let S be a set of attribute names, defined as: $S = \{a \mid a \in R(A) \wedge type(a) = \text{string}\}$, *i.e.*, $S \subseteq R(A)$. Let *meaning* be a function that receives a word, looks-up for its meaning in a dictionary, and returns it. There are *synonyms* in attribute $a \in S$ *iff*: $\exists\ t_1, t_2 \in r : v(t_1,a) \neq v(t_2,a) \wedge meaning(v(t_1,a)) = meaning(v(t_2,a))$.

  Example: The attribute *Occupation* contains the values *Professor* and *Teacher* in different tuples, which in fact represent the same occupation.

- **Violation of business domain constraint**

  Definition: Let *check* be a function that receives the set of all values of an attribute, checks whether a given constraint is respected, and returns a boolean value. There is a *violation of business domain constraint* in attribute $a \in R(A) : check(v(T,a)) = false$.

  Note: As defined in section 2.1, $v(T,a)$ represents the values that attribute a take for all tuples.

  Example: The values of attribute *Invoice_Date* must appear in the relation by ascending order, but this does not happens.

2.2.3 Multiple Attributes of a Single Tuple

The following DQ problems were identified by analyzing the values of multiple attributes of a single tuple, as illustrated in Figure 4.

- **Semi-empty tuple**

  Definition: Let θ be a user-defined threshold (a real number between 0 and 1), and S the set of attribute names that are empty in tuple t, defined as: $S = \{a \mid a \in R(A) \wedge v(t,a) = null\}$, *i.e.*, $S \subseteq R(A)$. Let m be the cardinality of set S, defined as: $m = |S|$, and n be the relation schema degree. The tuple t is a *semi-empty tuple iff*: $m/n \geq \theta$.

Example: If 60% or more of the tuple attributes are empty, then the tuple is classified as semi-empty

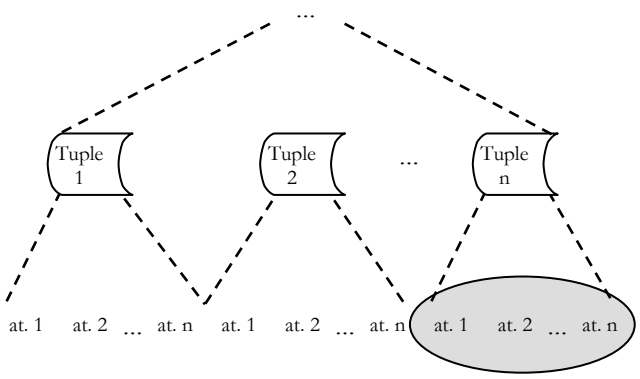

Figure 4: Multiple attributes of a single tuple

- **Violation of functional dependency**

  Definition: Let a_2 be an attribute whose value functionally depends on the values of other attributes. The set of these attribute names is defined as: $S = \{a_1 \mid a_1, a_2 \in R(A)$: the value of a_2 functionally depends on the value of $a_1\}$, *i.e.*, $S \subseteq R(A)$. Let *value* be a function that receives a set of values of a tuple, computes the value of the functional dependent attribute, and returns it. There is a *violation of functional dependency* in tuple t *iff*: $\exists\ t \in r$: $value(v(t,S)) \neq v(t,a_2)$.

  Example: There is a functional dependency among *Zip_Code* and *City*. Each value of the first attribute must be associated with exactly one value of the second. Therefore, the following values of two customer tuples violate the functional dependency: (*Zip_Code = 4000*; *City =* "*Porto*") and (*Zip_Code = 4000*; *City =* "*Lisboa*").

- **Violation of business domain constraint**

  Definition: Let *check* be a function that receives the set of values of a tuple, checks whether a given constraint x is respected, and returns a boolean value. Let S be a set of attribute names, defined as: $S = \{a \mid a \in R(A) \wedge a$ is used in the formulation of $x\}$, *i.e.*, $S \subseteq R(A)$. There is a *violation of business domain constraint* in tuple $t \in r$ *iff*: $check(v(t,S)) = false$.

  Note: As defined in section 2.1, $v(t,S)$ represents the values in tuple t of the attributes that belong to S.

  Example: The business domain constraint among attribute values: *Total_Product = Quantity * Sell_Price*, does not hold for a given tuple of the *Sales_Details* relation.

2.3 DQ Problems at the Level of a Single Relation

The DQ problems described in this section were identified by analyzing the values of multiple attributes in multiple tuples of a relation, as illustrated in Figure 1.

- **Approximate duplicate tuples**

  Definition: Let S be a set of attribute names, defined as: $S = \{a \mid a \in R(A) \wedge a$ does not belong to the primary key$\}$, *i.e.*, $S \subseteq R(A)$. Let θ be a real number between 0 and 1. Let *similarity* be a function that receives two values of an attribute, computes the similarity among them, and returns it (also as a real number between 0 and 1). There are *approximate duplicate tuples* in relation r *iff*: $\exists$

$t_1, t_2 \in r \;\; \forall\, a \in S : similarity(v(t_1,a),v(t_2,a)) \geq \theta \wedge t_1 \neq t_2$.

Example: The tuple *Customer*(10, '**Smith Barney**', '**Flowers Street, 123**', 502899106) is an approximate duplicate of the tuple *Customer*(72, '**S. Barney**', '**Flowers St., 123**', 502899106).

- **Inconsistent duplicate tuples**

  Definition: Let S be a set of attribute names, defined as: $S = \{a \mid a \in R(A) \wedge a$ does not belong to the primary key$\}$, *i.e.*, $S \subseteq R(A)$. Let θ be a real number between 0 and 1. Let *similarity* be a function that receives two values of an attribute, computes the similarity between them, and returns it (also as a real number between 0 and 1). There are *inconsistent duplicate tuples* in relation r *iff*: $\exists\, a_2 \in S, t_1, t_2 \in r \;\; \forall\, a_1 \in S \backslash \{a_2\} : similarity(v(t_1,a_1),v(t_2,a_1)) \geq \theta \wedge similarity(v(t_1,a_2), v(t_2,a_2)) < \theta$.

  Example: The tuple *Customer*(10, 'Smith Barney', '**Flowers Street, 123**', 502899106) is an inconsistent duplicate of the tuple *Customer*(72, 'Smith Barney', '**Sun Street, 321**', 502899106).

- **Violation of business domain constraint**

  Definition: Let *check* be a function that receives the attribute values of all tuples, checks whether a given constraint x is respected, and returns a boolean value. Let S be a set of attribute names, defined as: $S = \{a \mid a \in R(A) \wedge a$ is used in the formulation of $x\}$, *i.e.*, $S \subseteq R(A)$. There is a *violation of business domain constraint* in tuple t *iff*: $check(v(T,S)) = false$.

  Note: $v(T,S)$ represents the values that the attributes belonging to S take for all tuples.

  Example: The maximum number of products families allowed in relation *Products_Families* is 10, but the existent number of families is 12.

2.4 DQ Problems at the Level of Multiple Relations

In this section, we present the DQ problems detected when analyzing the values from multiple relations, as presented in figure 1.

We assume there is a relationship among the relation schemas $R_1(A_1)$ and $R_2(A_2)$ of a data source *DS*. Let S and T be sets of attribute names, defined as: $S = \{a \mid a \in R_1(A_1) \wedge a$ belongs to the foreign key that establishes the relationship with $R_2(A_2)\}$, *i.e.*, $S \subseteq R_1(A_1)$, and $T = \{a \mid a \in R_2(A_2) \wedge a$ belongs to the primary key$\}$, *i.e.*, $T \subseteq R_2(A_2)$. These two sets are used in the following definitions.

- **Referential integrity violation**

  Definition: Let V be the set of values of the primary key attributes, defined as: $V = \{v(t,T) \mid t \in r_2\}$. There is a *referential integrity violation* among relations r_1 and r_2 *iff*: $\exists\, t \in r_1 : v(t,S) \notin V$.

  Example: The attribute *Customer_Zip_Code* of the *Customer* relation contains the value *5100*, which does not exists in the *Zip_Code* relation.

- **Incorrect reference**

  Definition: Let V be the set of values of the primary key attributes, defined as: $V = \{v(t,T) \mid t \in r_2\}$. Let $u(t,S)$ be the correct and updated value that was supposed to be in the foreign key S of tuple t of relation r_1. There is an *incorrect reference* among relations r_1 and r_2 *iff*: $\exists\, t \in r_1 : v(t,S) \in V \wedge v(t,S) \neq u(t,S)$.

  Example: The attribute *Customer_Zip_Code* of the *Customer* relation contains the value *4415*, instead of *4445*; both zip codes exist in the *Zip_Code* relation.

- **Heterogeneity of syntaxes**

  Definition: Let $G(a)$ be the syntax of attribute a, given by a grammar or a regular expression. There is a *heterogeneity of syntaxes* among relations r_1 and r_2 *iff*: $\forall\ a_1 \in R_1(A_1)$, $a_2 \in R_2(A_2)$: $type(a_1) = type(a_2) \wedge G(a_1) \neq G(a_2)$.

  Example: The attribute *Order_Date* of relation *Orders* has the syntax *dd/mm/yyyy*, while the attribute *Invoice_Date* of relation *Invoices* has the syntax *yyyy/mm/dd*.

- **Circularity among tuples in a self-relationship**

  Definition: Let U be a set of attribute names, defined as: $U = \{a \mid a \in R_1(A_1) \wedge a$ belongs to the primary key$\}$, *i.e.*, $U \subseteq R_1(A_1)$. Let V be the set that contains the primary key values of all existing tuples in r_1, defined as: $V = \{v(t,U) \mid t \in r_1\}$. Let v be the value of a primary key: $v \in V$. Let W be the set that, starting from the tuple identified by the primary key v, contains the foreign key values of all other tuples related with it, defined as: $W = \{v(t_1,S) \mid v(t_1,S) = v(t_2,U) \wedge t_1, t_2 \in r_1\}$. There is a *circularity among tuples in a self-relationship* in relation r_1 *iff*: $v \in W$.

  Example: A product may be a sub-product in another product and this information is stored in attribute *Sub-product_Cod* of the product; In relation *Products* there exists the information that product *X* (*Product_Cod* = '*X*') is sub-product of *Y* (*Sub-product_Cod* = '*Y*') and simultaneously that product *Y* (*Product_Cod* = '*Y*') is sub-product of *X* (*Sub-product_Cod* = '*X*'); this is an impossible situation.

- **Violation of business domain constraint**

  Definition: Let *check* be a function that receives the attribute values of the tuples from relations r_1 and r_2, checks whether a given constraint x is respected, and returns a boolean value. Let U and V be sets of attribute names, defined as: $U = \{a \mid a \in R_1(A_1) \wedge a$ is used in the formulation of $x\}$, *i.e.*, $U \subseteq R_1(A_1)$, and $V = \{a \mid a \in R_2(A_2) \wedge a$ is used in the formulation of $x\}$, *i.e.*, $V \subseteq R_2(A_2)$. Let W and Z be sets of attribute values of related tuples from each relation, defined as: $W = \{v(t_1,U) \mid v(t_1,S) = v(t_2,T) \wedge t_1 \in r_1 \wedge t_2 \in r_2\}$ and $Z = \{v(t_2,V) \mid v(t_2,T) = v(t_1,S) \wedge t_1 \in r_1 \wedge t_2 \in r_2\}$. There is a *violation of business domain constraint* among relations r_1 and r_2 *iff*: $check(W, Z) = false$.

  Example: The attribute *Invoice_Total* of a tuple of the relation *Invoices* contains the value *100*, while the sum of the values of attribute *Product_Value* (for each product of the invoice) of the relation *Invoices_Details* is only equal to *90* (instead of *100*).

2.5 DQ Problems at the Level of Multiple Data Sources

The DQ problems presented below were identified by analyzing the values of multiple data sources, as illustrated in Figure 1. As referred in section 1, this paper only addresses the DQ problems related with the instances (values) of data, *i.e.*, the extensional level [6]. There are other kinds of DQ problems that occur at the intensional level, *i.e.*, problems related with the structure of data [6], also known as problems among data schemas. For the reader interested in these problems, we suggest the work of Kashyap and Sheth [5].

In this section, we assume that the relation schemas $R_1(A_1)$ and $R_2(A_2)$ belong to two different data sources, respectively, DS_1 and DS_2. Both schemas concern the same real-world entity (*e.g.*, customers). We also assume that relation schema heterogeneities among DS_1 and DS_2 are solved, *i.e.*, two attributes referring to the same real-world property (*e.g.*, *unitary price*) have the same name. However, the number of attributes used in each data schema may be different.

- **Heterogeneity of syntaxes**

  Definition: Let $G(a)$ be the syntax of attribute a, given by a grammar or a regular expression. There is *heterogeneity of syntaxes* among relations r_1 and r_2 *iff* : $\forall\, a_1 \in R_1(A_1)$, $a_2 \in R_2(A_2)$: $a_1 = a_2 \wedge type(a_1) = type(a_2) \wedge G(a_1) \neq G(a_2)$.

  Example: The attribute *Insertion_Date* of relation *Customers* from $\mathcal{DS}_1$ has the syntax *dd/mm/yyyy*, while the attribute *Insertion_Date* of relation *Customers* from $\mathcal{DS}_2$, has the syntax *yyyy/mm/dd*.

- **Heterogeneity of measure units**

  Definition: Let S be the set of attribute names common to both relations, defined as: $S = \{a \mid a \in R_1(A_1) \wedge a \in R_2(A_2) \wedge type(a) = \text{numeric}\}$. Let k be a numeric constant value. There is *heterogeneity of measure units* in attribute a of relations r_1 and r_2 *iff*: $\exists\, a \in S \;\; \forall\, t_1 \in r_1 \;\; \exists\, t_2 \in r_2$: $v(t_1,a) = k * v(t_2,a) \wedge k > 0 \wedge k \neq 1$.

  Example: The attribute *Product_Sell_Price* is represented in euros in DS_1, while in DS_2 is represented in dollars.

- **Heterogeneity of representation**

  Definition: Let S be the set of attribute names common to both relations, defined as: $S = R_1(A_1) \cap R_2(A_2)$. Let *translate* be a function that receives an attribute value from a relation, looks-up in a dictionary (lookup table) for the corresponding value in the other relation, and returns it. There is *heterogeneity of representation* in attribute a among relations r_1 and r_2 *iff*: $\exists\, a \in S$, $t_1 \in r_1$, $t_2 \in r_2$: $v(t_1,a) \neq v(t_2,a) \wedge translate(v(t_1,a)) = v(t_2,a)$.

  Example: To represent the attribute *Gender* the values *F* and *M* are used in DS_1, while in DS_2 are used the values *0* and *1*.

- **Existence of synonyms**

  Definition: Let S be the set of attribute names common to both relations, defined as: $S = \{a \mid a \in R_1(A_1) \wedge a \in R_2(A_2) \wedge type(a) = \text{string}\}$. Let *meaning* be a function that receives a word, looks-up for its meaning in a dictionary, and returns it. There are *synonyms* in attribute a among relations r_1 and r_2 *iff*: $\exists\, a \in S$, $t_1 \in r_1$, $t_2 \in r_2$: $meaning(v(t_1,a)) = meaning(v(t_2,a)) \wedge v(t_1,a) \neq v(t_2,a)$.

  Example: The relation *Occupations* of DS_1 contains a tuple with *Professor*, while the equivalent relation in DS_2 contains a tuple with *Teacher*; both represent the same occupation.

- **Existence of homonyms**

  Definition: Let S be the set of attribute names common to both relations, defined as: $S = \{a \mid a \in R_1(A_1) \wedge a \in R_2(A_2) \wedge type(a) = \text{string}\}$. Let $meaning_1$ and $meaning_2$ be functions that receive a word, look-up for its meaning in a dictionary (in the context of DS_1 or DS_2), and return it. There are *homonyms* in attribute a among relations r_1 and r_2 *iff*: $\exists\, a \in S$, $t_1 \in r_1$, $t_2 \in r_2$: $meaning_1(v(t_1,a)) \neq meaning_2(v(t_2,a)) \wedge v(t_1,a) = v(t_2,a)$.

  Example: In relation *Products* of DS_1, there exists a product named *Mouse* (a branch of a company sells computer hardware), while in relation *Products* of DS_2, there also exists a product named *Mouse* (another branch of the company sells domestic animals, so the products here are the animals themselves).

- **Approximate duplicate tuples**

  Definition: Let S be a set of attribute names, defined as: $S = \{a \mid a \in R(A) \wedge a$ does not belong to the primary key$\}$, *i.e.*, $S \subseteq R(A)$. Let θ be a real number between 0 and 1. Let *similarity* be a

function that receives two values of an attribute, computes the similarity between them, and returns it (also as a real number between 0 and 1). There are *approximate duplicate tuples* among relations r_1 and r_2 *iff*: $\exists\ t_1 \in r_1, t_2 \in r_2\ \ \forall\ a \in S : similarity(v(t_1,a),v(t_2,a)) \geq \theta$.

Example: The tuple *Customer*(10, '**Smith Barney**', '**Flowers Street, 123**', 502899106) in DS_1 is an approximate duplicate of the tuple *Customer*(27, '**Smith B.**', '**Flowers St., 123**', 502899106) in DS_2.

- **Inconsistent duplicate tuples**

  Definition: Let S be a set of attribute names, defined as: $S = \{a \mid a \in R(A) \wedge a$ does not belong to the primary key$\}$, *i.e.*, $S \subseteq R(A)$. Let θ be a real number between 0 and 1. Let *similarity* be a function that receives two values of an attribute, computes the similarity between them, and returns it (also as a real number between 0 and 1). There are i*nconsistent duplicate tuples* among relations r_1 and r_2 *iff*: $\exists\ t_1 \in r_1,\ t_2 \in r_2,\ a_2 \in S\ \ \forall\ a_1 \in S\backslash\{a_2\}$: $similarity(v(t_1,a_1),v(t_2,a_1)) \geq \theta \wedge similarity(v(t_1,a_2),v(t_2,a_2)) < \theta$.

  Example: The tuple *Customer*(10, 'Smith Barney', '**Flowers Street, 123**', 502899106) in DS_1 is an inconsistent duplicate of the tuple *Customer*(27, 'Smith Barney', '**Sun Street, 321**', 502899106) in DS_2.

- **Violation of business domain constraint**

  Definition: Let *check* be a function that receives the attribute values of the tuples from relations r_1 and r_2 (of DS_1 and DS_2), checks whether a given constraint x is respected, and returns a boolean value. Let S be the set of attribute names common to both relations, defined as: $S = \{a \mid a \in R_1(A_1) \wedge a \in R_2(A_2) \wedge a$ is used in the formulation of $x\}$. Let T and U be sets that contain the attribute values of all tuples from relations r_1 and r_2, defined as: $T = \{v(t_1,S) \mid t_1 \in r_1\}$ and $U = \{v(t_2,S) \mid t_2 \in r_2\}$. There is a *violation of business domain constraint* among relations r_1 and r_2 *iff*: $check(T, U) = false$.

  Example: The maximum number of products families allowed is 10; the relation *Product_Families* in DS_1 contains 7 families, and the relation *Product_Families* in DS_2 contains 8 families; the number of distinct product families resulting from the integration (union) of both sources is 11; this number violates the constraint.

2.6 Summary

Table 1 presents a summary of our taxonomy of DQ problems. The problems and the corresponding granularity levels where they occur are shown in the table.

3. RELATED WORK

Kim *et al*. [7] present a quite complete taxonomy of DQ problems, describing the logic behind its structure. They adopt a successive hierarchical refinement approach. The taxonomy is based on the premise that DQ problems manifest in three different ways: missing data; not missing but wrong data; and not missing and not wrong but unusable. Unusable data occurs when two or more databases are integrated or representation standards are not consistently used when entering data. The taxonomy is a hierarchical decomposition of these three basic manifestations of DQ problems. Considering the approach used and the DQ problems identified, this taxonomy is the closest to ours.

Data Quality Problem	Attribute/Tuple			Single Relation	Multiple Relations	Mult. Data Sources
	Attrib.	Column	Row			
Missing value	x					
Syntax violation	x					
Incorrect value	x					
Domain violation	x					
Invalid substring	x					
Misspelling error	x					
Imprecise value	x					
Violation of business domain constraint	x	x	x	x	x	x
Unique value violation		x				
Existence of synonyms		x				x
Semi-empty tuple			x			
Violation of functional dependency			x			
Approximate duplicate tuples				x		x
Inconsistent duplicate tuples				x		x
Referential integrity violation					x	
Incorrect reference					x	
Heterogeneity of syntaxes					x	x
Circularity among tuples in a self-relationship					x	
Heterogeneity of measure units						x
Heterogeneity of representation						x
Existence of homonyms						x

Table 1: DQ problems organized by granularity level

Müller and Freytag [9] roughly classify DQ problems into syntactical, semantic, and coverage anomalies. Syntactical anomalies describe characteristics concerning the syntax and values used for representation of the entities (*e.g*. lexical errors, domain syntax errors). Semantic anomalies hinder the data collection from being a comprehensive and non-redundant representation of the real-world (*e.g*. duplicates, contradictions). Coverage anomalies are related with the amount of entities and entities properties from the real-world actually stored in the data collection (*e.g*. missing values). This work is limited to DQ problems that occur in a single relation of a single source, so important DQ problems are not covered.

Rahm and Do [14] distinguish between single-source and multi-source problems as we do. However, at single-source they do not divide the problems into those that occur in a single relation and those that occur as a result of existing relationships among multiple relations. Single-source and multi-source problems are divided into schema-related and instance-related problems. Schema-related problems are those that can be addressed by improving the schema design, schema translation and schema integration. Instance-related problems correspond to errors and inconsistencies in the actual data contents that can not be prevented at the schema level. As referred in the introduction, we are only concerned with the DQ problems related with the instances of the data, so we do not make this separation. In single source problems, for both schema-related and instance-related, they distinguish between the following scopes: attribute; record; record type and source. This is similar to the organization that we present in our taxonomy.

Even tough the term used may be different (*e.g.* enterprise constraint violation; business rule violation), the DQ problem *violation of business domain constraint* included in our taxonomy is mentioned in almost every book about databases (*e.g.* [2, 16]). However, surprisingly it is not included in any of the taxonomies analysed. The other new DQ problems also introduced by our taxonomy are: (*i*) semi-empty tuple; (*ii*) heterogeneity of syntaxes (at the level of multiple relations and multiple data sources); and (*iii*) circularity among tuples in a self-relationship. All the problems identified in the three taxonomies are also covered by ours, although the names used to label the problems are sometimes different. Finally, the DQ problems have been described only through a textual description, while we present a rigorous definition for each problem.

4. CONCLUSION

This paper has presented our taxonomy of DQ problems. The taxonomy results from the research conducted to identify DQ problems on each granularity level of the usual data organization model. The study followed a bottom-up approach, from the lowest (attribute/tuple) to the highest granularity level (multiple data sources) where DQ problems may appear. The taxonomy was also presented in the paper following that approach. Six groups of related DQ problems were derived from the four granularity levels. As the approach followed to identify the problems was exhaustive and systematic, it allows us to be confident that no other problem is missing.

The DQ problems included in our taxonomy were specified through rigorous definitions. This feature distinguishes our taxonomy from the related ones, since they only use text to describe the problems. We believe that giving a formal framework to DQ problems is a valuable contribution, since: (*a*) it is the only way to assure a clear and precise definition for each DQ problem; and (*b*) it is useful because it specifies what is required to detect automatically the problem, *i.e.*: (*i*) the metadata knowledge needed; (*ii*) the mathematical expression that defines the DQ problem, which can be seen as a logical procedure (rule) to detect it; and (*iii*) eventually the function that is required to perform some transformation. These elements are explicitly included in each definition of DQ problem.

This work is a first step towards the development of a tool to automatically detect DQ problems. The entire set of DQ problems that may affect data is now known and understood by us. We also know what is needed to detect each DQ problem and how that can be translated to a computational method. All these items need to be organized to produce the DQ tool architecture. In fact, this is what we intend to do as our next work. After, we intend to start the tool development. We believe that it will complement the limited detection capabilities currently supported by commercial data profiling tools.

ACKNOWLEDGMENTS

We would like to thank Helena Galhardas for the fruitful discussions and useful comments that helped us to improve the contents of the paper.

REFERENCES

[1] Atzeni, P. and Antonellis, V. – *Relational Database Theory*. The Benjamin/Cummings Publishing Company, Inc., 1983.

[2] Connolly, T. and Begg, C. – *Database Systems: A Practical Approach to Design, Implementation and Management*. Addison Wesley Longman Limited, 1999. ISBN 0-201-34287-1.

[3] Dasu, T.; Vesonder, G. T. and Wright, J. R. – "Data Quality through Knowledge Engineering". In *Proceedings of the SIGKDD'03 Conference*, Washington. August 2003. pp. 705-710.

[4] Galhardas, H.; Florescu, D.; Shasha, D.; Simon, E. and Saita, C.-A. – "Data Cleaning: Language, Model, and Algorithms". In *Proceedings of the Very Large Databases Conference* (VLDB). 2001.

[5] Kashyap, V. and Sheth, A. – "Schematic and Semantic Similarities Between Database Objects: a Context-Based Approach". *Very Large Databases Journal*, 5 (4). 1996. pp. 276–304.

[6] Kedad, Z. and Métais E. – "Ontology-Based Data Cleaning". *Lecture Notes in Computer Science*, 2553, 2002. pp. 137 – 149.

[7] Kim, W.; Choi, B.-J.; Hong, E.-K.; Kim, S.-K. and Lee, D. – "A Taxonomy of Dirty Data". *Data Mining and Knowledge Discovery*, 7. 2003. pp. 81-99.

[8] Meta Group – Data Warehouse Scorecard. Meta Group, 1999.

[9] Müller, H. and Freytag, J.-C. – "Problems, Methods, and Challenges in Comprehensive Data Cleansing". *Technical Report HUB-IB-164*, Humboldt University, Berlin, 2003.

[10] Oliveira, P.; Rodrigues, F. and Henriques, P. – "Limpeza de Dados: Uma Visão Geral". In Belo, O.; Lourenço, A. and Alves, R. (Eds.) – *Proceedings of Data Gadgets 2004 Workshop – Bringing Up Emerging Solutions for Data Warehousing Systems* (in conjunction with JISBD'04), Málaga, Spain, November 2004. pp. 39-51 (in Portuguese).

[11] Oliveira, P.; Rodrigues, F.; Henriques, P. and Galhardas, H. – "A Taxonomy of Data Quality Problems". In *Proceedings of the 2nd International Workshop on Data and Information Quality* (in conjunction with CAiSE'05), Porto, Portugal, June 2005.

[12] Orr, K. – "Data Quality and Systems Theory". *Communications of the ACM*, 41 (2). 1998. pp. 66-71.

[13] Pipino, L.; Lee, Y. and Wang, R. – "Data Quality Assessment". *Communications of the ACM*, 45 (4). 2002. pp. 211-218.

[14] Rahm, E. and Do, H. H. – "Data Cleaning: Problems and Current Approaches". *IEEE Bulletin of the Technical Committee on Data Engineering*, 24 (4). 2000.

[15] Sattler, K. and Schallehn, E. – "A Data Preparation Framework based on a Multidatabase Language". In *Proceedings of International Database Engineering and Applications Symposium (IDEAS 2001)*, Grenoble, France. IEEE Computer Society. 2001. pp. 219-228.

[16] Ullman, J. and Widom, J. – *A First Course in Database Systems*. Prentice-Hall, Inc., 1997. ISBN 0-13-861337-0.

[17] Wand, Y. and Wang, R. – "Anchoring Data Quality Dimensions in Ontological Foundations". *Communications of the ACM*, 39 (11). 1996. pp. 86-95.