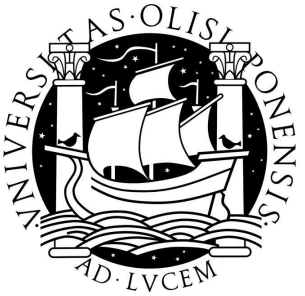
Explicitly Involving the User in a Data Cleaning Process

Helena Galhardas, Antónia Lopes, Emanuel Santos

DI–FCUL–TR–2010–03

DOI:http://hdl.handle.net/10455/6674

February 14, 2011



Published at Docs.DI (http://docs.di.fc.ul.pt/), the repository of the Department of Informatics of the University of Lisbon, Faculty of Sciences.

Explicitly Involving the User in a Data Cleaning Process

Helena Galhardas [#1], Antónia Lopes [*], Emanuel Santos [#3]

[#] *INESC-ID and Technical University of Lisbon*
Av. Prof. Dr. Aníbal Cavaco Silva 2744-016 Porto Salvo, Portugal
[1] hig@inesc-id.pt
[3] esantos@ist.utl.pt

[*] *University of Lisbon*
FCUL, Campo Grande 1749-016 Lisboa, Portugal
mal@di.fc.ul.pt

February 14, 2011

Abstract

Data cleaning and Extract-Transform-Load processes are usually modeled as graphs of data transformations. These graphs typically involve a large number of data transformations, and must handle large amounts of data. The involvement of the users responsible for executing the corresponding programs over real data is important to tune data transformations and to manually correct data items that cannot be treated automatically.

In this paper, we extend the notion of data cleaning graph in order to better support the user involvement in data cleaning processes. We propose that data cleaning graphs include: (i) *data quality constraints* to help users to identify the points of the graph and the records that need their attention and (ii) *manual data repairs* for representing the way users can provide the feedback required to manually clean some data items. We provide preliminary experimental results that show, for a real-world data cleaning process, the significant gains obtained with our approach in terms of the quality of the data produced and the cost incurred by users in data visualization and updating tasks.

1 Introduction

Data cleaning and ETL (Extraction, Transformation, and Loading) processes are commonly modeled as workflows or graphs of data transformations. There is a large variety of commercial tools that support the specification and execution of those graphs. Some were built specifically for this purpose, such as *Informatica*[1] or *Talend*[2], and others are Relational Database Management Systems

[1] http://www.informatica.com
[2] http://www.talend.com

(RDBMS) extensions, such as *SQL Server 2008 Integration Services*[3].

The logic underlying real-world data cleaning processes is usually quite complex. These processes often involve tens of data transformations that are implemented, for instance, by pre-defined operators of the chosen ETL tool, SQL scripts, or procedural code. Moreover, these processes have to deal with large amounts of input data. Therefore, as pointed out by Rahm and Do in [21], in general, it is not easy to devise a graph of data transformations able to always produce accurate data according to the data cleaning specific requirements. This happens for two main reasons: *(i)* individual data transformations that consider all possible data quality problems are difficult to write; and/or *(ii)* a fully automated solution that meets the quality requirements is not always attainable. The first reason implies that the underlying logic of each individual transformation often needs to undergo several revisions. In particular, this is likely to happen when the cleaning process is executed over a new batch of data. Hence, it is important that users responsible for executing the data cleaning processes have adequate support for tuning data transformations. The second reason implies that, in general, a portion of the cleaning work has to be done manually and, hence, it is important to also support the user involvement in this activity. In the sequel, we denote the first type of action over a data cleaning process as *data transformation tuning* and the second one as *manual data repair*.

The problem is that, when using ETL and data cleaning tools, intermediate results obtained after individual data transformations are typically not available for inspection or eventual manual correction — the output of a data transformation is directly pipelined into the input of the transformation that follows in the graph. The solution we envisage for this problem is to support the specification of the points in the graph of data transformations where intermediate results must be available, together with the *quality constraints* that this data should meet, if the upward data transformations correctly transform all the data records as expected. Because assignment of blame is crucial for identifying where the problem is, the records responsible for the violation of quality constraints are highlighted. This information is useful both for tuning data transformations that do not handle the data as expected and for performing the manual cleaning of records not handled automatically by data transformations.

While the tuning of data transformations requires some knowledge about the logic of the cleaning process, it is useful that manual data repairing actions can also be performed in a black-box manner, for instance by application end-users. As already advocated in the context of Information Extraction [5], in many situations, data consumers have knowledge about how to correctly handle the rejected records and, hence, can provide critical feedback into the data cleaning program. Our proposal is that the developer of the cleaning process has the ability to specify, in specific points of the graph of data transformations where intermediate results are available, the way users can provide the feedback required to manually clean certain data items. This may serve two different purposes: for guiding the effort of the user that is executing the cleaning process (even if he/she has some knowledge about the underlying logic) and for supporting the feedback of users that are just data consumers.

[3] http://www.microsoft.com/sqlserver/2008/en/us/integration.aspx

tId	Full Name	tName
1	Luis Carriço	Carriço, L.
2	André Leal Santos	Santos, A. L.
3	André Santos	Santos, A.
4	Antónia Lopes	Lopes, A.
5	Marco Sá	Sá, M.
6	Carlos Teixeira	Teixeira, C.

Figure 1: Team table: tId is a sequential number that uniquely identifies each member; FullName and tName store, respectively, the full and abbreviated name of the team member.

1.1 Motivating example

Nowadays, the availability of information about the impact and relevance of research results is crucial. In fact, the analysis of the number of publication citations is commonly used as a means to evaluate individuals and research institutions. Several on-line tools exist with the purpose of retrieving citations for a given person (e.g., *Google Scholar*[4] and *CiteSeer*[5]). More comprehensive tools, such as *Microsoft Academic Search*[6], also provide the values of several citation metrics such as g-index [9] and h-index [15].

Although these tools are very useful and powerful, they are not enough when the goal is the evaluation of the research performance of a group of people — research group, lab, institution, or country. In this case, after gathering the information about the publications of individuals, additional computation is required. Moreover, it is necessary to deal with several data quality problems, namely those related with the existence of *homonyms* — different researchers that use the same name, and *synonyms* — different variants of the same name.

Let us consider that the information required for computing the research performance metrics for a given team is collected into a database with tables Team and Pub as illustrated in Figures 1 and 2 (this is in fact a simplification of the real database used in the CIDS system [7]). The Team table is manually filled with accurate information about the team members. The Pub table stores the information about the citations of team members obtained through queries posed to Google Scholar.

The relationship that exists between the two tables, through the foreign key tId, associates all the publications, and corresponding information, to a team member. However, this association might not be correct, namely due to the existence of homonyms. In our example, the member in Team named "Luis Carriço" refers to a colleague of us at FCUL and the Pub record with pid 4 is not, in fact, authored by him, but by a homonym. Another data quality problem that affects these two tables is the multitude of variants that author names admit, specially those that use letters with diacritical marks. For instance, the records of Pub shown in Figure 2 contain two synonyms of "Carriço, L." — "Carrico, L." and "Carri{c{c}} o, L.", and three synonyms of "Sá, M." — "de Sa, M.", "Sa, M." and "S{\\' a}, M.".

Clearly, the computation of reliable research performance indicators for a group of researchers requires a data cleaning process that, among other things,

[4]http://scholar.google.com/
[5]http://citeseer.ist.psu.edu/
[6]http://academic.research.microsoft.com/

pId	tId	title	authors	year	event	link	cits	citsNs
1	1	Users and usage driven adaptation of digital talking books	Duarte, C. and Carri{\c{c}}o, L.	2005	International Conference on Human Computer Interaction	scholar?cluster=17494767326604985714	12	2
2	1	Ubiquitous Psychotherapy	Sa, M. and Carrico, L. and Antunes, P.	2007	IEEE Pervasive Computing	scholar?cluster=17926891015808149598	10	1
3	1	Ubiquitous Psychotherapy	de Sa, M. and Carrico, L. and Antunes, P.	2007	IEEE Pervasive Computing	scholar?cluster=7857184169753406012	9	6
4	1	Reduction of the 2, 4, 6-trichloroanisole content in cork stoppers using gamma radiation	Pereira, C. and Gil, L. and Carrico, L.	2007	Radiation Physics and Chemistry	scholar?cluster=7369389612736854711	15	5
5	2	Managing duplicates in a web archive	Gomes, D. and Santos, A. L. and Silva, M. J.	2006	ACM Symposium on Applied Computing	scholar?cluster=12453222090111315 4	17	9

Figure 2: Pub table: pId is the unique identification of each answer returned by the search engine; tId is a foreign key to the Team table and identifies the team member associated to the current answer; title is the title of the publication; authors is a string composed by the names of the authors of the publication according to the format (last-name1, first-name-initials1 and last-name2, first-name-initials2 and...); year is the year of the publication; event is the name of the conference or journal; link stores the search engine link of the publication location; cits and citsNs are the number of citations and non-self citations, respectively.

deals with the problems of synonyms and homonyms pointed before. The Team table can be used as reference to identify and correct these problems. State-of-art procedures to solve synonyms are based on the use of approximate string matching [14]. Names as "Carriço, L." in tuple 1 of Team table and "Carrico, L." in tuple 2 of Pub table can easily be found as matches. However, it may also happen that these procedures find several possible correct names for the same author name. For example, "Santos, A." and "Santos, A. L." are the names of two team members and both match the author name "Santos, A. L." encountered in the authors field of tuple 5 in the Pub table. By other words, the two pairs of names ("Santos, A.", "Santos, A. L.") and ("Santos, A. L.", "Santos, A. L.") are similar enough so that both entries of the Team table are considered as potential candidates of team member names for "Santos, A. L.". The problem that remains to be solved is which of the two to choose, or to decide if none of them does in fact correspond to the individual "Santos, A. L.". Given the available information, we believe that this kind of domain knowledge can only be brought by a user that is aware of the team members and the corresponding research work. The syntactic similarity value that exists between the two pairs of strings is not enough for automatically taking this decision.

The detection of homonyms in the context of person names has also been object of active research. For instance, [17] has shown that the detection of homonyms among author names can benefit from the use of knowledge about co-authorship. If information about co-authors of every given author is available, then a clustering algorithm can be applied with the purpose of putting into the same cluster those author names that share a certain amount of co-authors. In principle, the author names that belong to the same cluster most probably correspond to the same real entity. The problem that remains is how to obtain accurate co-authorship information. Clearly, automatic methods for calculating this information from publications are also subject to the problem of homonyms and, hence, the produced information in general is not accurate [17]. In this case, we believe that the circularity of the problem can only be broken by involving the user in the cleaning of the co-authorship information that was automatically

obtained.

1.2 Our proposal

The example just presented shows the importance of being able to automatically transforming and cleaning data while efficiently employing user's efforts to overcome the problems that were not possible to handle automatically. In this paper, we propose a way of incorporating the user involvement in these processes and present a modeling primitive — the *data cleaning graph*, that supports the description of data cleaning processes that are conceived having user involvement in mind.

A data cleaning graph encloses a graph of data transformations as used, for instance, in [25] and [13]. The output of each transformation is explicitly expressed and associated with a *quality constraint*. This constraint expresses the criteria that data produced by the transformation should obey to and its purpose is to call the user attention for quality problems in the data produced by the transformation.

For instance, Figure 3 shows an excerpt of a data cleaning graph that was conceived for cleaning the Pub table so that the output table, CleanPub, has only publications authored by a member of Team. In this cleaning graph, the output of transformation T3 is subject to the quality constraint unique(aId, title). This is because, at this point, the idea is to have at most one matching team member, for each author of a publication in Pub. Because transformation T3 applies a string similarity function to decide if two names (one from the Pub table and the other from the Team table) are the same, it might happen that some data produced by T3 violates this constraint. For instance, due to the similarity function invoked in T3 and the corresponding threshold values imposed in our realization of the cleaning graph, both Team members *"Santos, A."* and *"Santos, A. L."* are found similar to Pub author *"Santos, A. L."* and this causes a violation of the quality constraint. The quality constraint will call the attention of the user to the tuples blamed for the violation.

Additionally, the data cleaning graph encloses the specification of the points where the manual data repairing actions may take place. The aim of this facility is to guide the intervention of the user (end-users included) and, hence, it is important to define which data records can be subject to manual modifications and how. We have only considered actions that can be applied to individual data records for repairing data. Three types of actions were found useful: remove a tuple, insert a tuple, and modify the values of certain attribute of a tuple.

Let us consider again the constraint violation associated to the output of transformation T3 caused by the pairs of strings *("Santos, A. L."; "Santos, A. L."), ("Santos, A. L."; "Santos, A.")*. Because, in this case, no automatic procedure is envisaged that is able to decide which of the two matching strings is the correct one (if any), user feedback was incorporated into the cleaning graph in the form of a *manual data repair*. The manual data repair Mdr3, associated with R3, consists of a possible delete action. This action is applied to the tuples produced by a view projecting the aId, aName, tName and title attributes of R3 and that includes only the records that are blamed for the constraint violation. In this example, the view projects almost all the attributes of the relation but we might use the view to exclude non relevant information and, in this way, limit the quantity of information the user has to process in order to decide which are

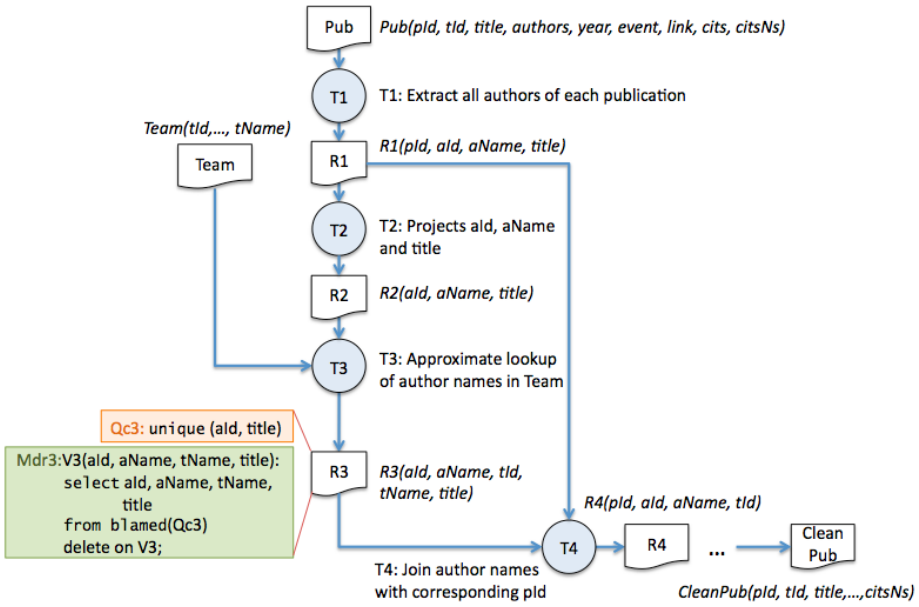


Figure 3: Excerpt of a data cleaning graph for cleaning Pub table

the appropriate manual data repairs to apply. In the case of the two tuples that violate Qc3, the user may be expert enough to be able to decide about them, without analyzing any other relation of the graph (in fact, this is the reason why the title of the publication was propagated until R3). The user can then delete one of the tuples generated by T3 that is involved in the violation of Qc3 — the tuple with title *"Managing duplicates in a web archive"* and aName *"Santos, A."*. By removing this tuple, the user wants to state that the author *"Santos, A. L."* of the corresponding publication is the team member named *"Santos, A. L."*.

1.3 Contributions

The main contributions of this paper are the following:

- The notion of data cleaning graph — a primitive that supports the modeling of data cleaning processes that explicitly define where and how user feedback is expected as well as which data should be inspected by the user.
- The operational semantics of the data cleaning graph that formally defines the execution of a data cleaning process over source data and past instances of manual data repairs. With this semantics, it is possible to interleave the tuning of data transformations with the manual data correction without requiring that the user repeats his/her feedback actions.
- Experimental results that show, for a real-world data cleaning application modeled as a data cleaning graph: (i) the gain in terms of the accuracy of the data produced, and (ii) the amount of user work involved, namely when compared with a totally manual data cleaning approach and with an automatic data cleaning solution without incorporating the user feedback.

1.4 Organization of the paper

This paper is organized as follows. Section 2 details our proposal, namely we define the notion of data cleaning graph and we explain the corresponding operational semantics. In Section 3, we present a case study of a data cleaning process and in Section 4, we report on the experimental results obtained that show the usefulness of our approach. In Section 5, we discuss the related work and in Section 6 we summarize the conclusions and future work.

2 Data cleaning graphs

In this section we present the concept of *data cleaning graph* — the modeling primitive we propose for describing data cleaning processes. We provide its operational semantics through an algorithm that manipulates sets of tuples.

Terminology. A domain D is a set of atomic values. We assume a set $\mathcal{D}$ of domains and a set $\mathcal{A}$ of names – attribute names – together with a function Dom: $\mathcal{A} \rightarrow \mathcal{D}$ that associates domains to attributes. We consider a set $\mathcal{R}$ of relations names and, for every $R \in \mathcal{R}$ a relation schema $sch(R)$ constituted by an ordered set $A_1, ..., A_n$ of attribute names. We write $R(A_1, ..., A_n)$. Given a relation schema $S = A_1, ..., A_n$, a S-tuple t is an element of $Dom(A_1) \times ... \times Dom(A_n)$. An *instance* of a relation R is a finite set of $sch(R)$-tuples.

We also consider a set $\mathcal{T}$ of data transformations. A *data transformation* $T \in \mathcal{T}$ consists of an ordered set $\mathbb{I}_T$ of input relation schemas, an output relation schema $\mathbb{O}_T$ and a total function that maps a sequence of $\mathbb{I}_T$-tuples to $\mathbb{O}_T$-tuples. We use $arity(T)$ to denote the number of schemas in $\mathbb{I}_T$ and $\mathbb{I}_T^i$ to denote the i-ary element of $\mathbb{I}_T$.

If G is a *direct acyclic graph* (DAG), we use $nodes(G)$ and $edges(G)$ to denote the set of nodes and edges. Moreover, we use ${}^\bullet n$ and $n^\bullet$ to denote, respectively, the sets $\{m : (m, n) \in edges(G)\}$ and $\{m : (n, m) \in edges(G)\}$. We also use $\leq_G$ to denote the partial order on the nodes of G, that is, $n \leq_G m$ iff there exists a directed path from n to m in G.

2.1 The notion of data cleaning graph

The notion of data cleaning graph builds on the notion of data transformation graph introduced in [13]. These graphs are tailored to relational data and include data transformations that can range from relational operators and extensions (like the mapper operator formalised in [4]) to procedural code. The partial order $\leq_G$ on the nodes of the graph G partially dictates the order of execution of the data transformations in the process. Data transformations not comparable can be executed in any order or even at the same time.

A data cleaning graph is a DAG, where nodes correspond to data transformations or relations, and edges connect (input and output) relations to data transformations. In order to support the user involvement in the process of data cleaning, we extend data transformations graphs with two kinds of labels: quality constraints and manual data repairs. On the one hand, each relation R in a cleaning graph is subject to a constraint expressing a *data quality criteria*. If the constraint is violated, it means that there is a set of tuples in the current instance of R that needs to be inspected by the user. Quality constraints can include the

traditional constraints developed for schema design, such as functional dependencies and inclusion dependencies, as well as constraints specifically developed for data cleaning, such as conditional functional dependencies [10]. The specific language used for specifying quality constraints is of no particular importance for this paper. We simply assume a fixed language $\mathcal{L}(\{R_1, ..., R_n\})$ for expressing constraints over a set of relations with its semantics defined in terms of a relation $\vDash$ between instances of $\{R_1, ..., R_n\}$ and sentences of $\mathcal{L}(\{R_1, ..., R_n\})$. However, we limit quality constraints to be monotonic in a way that is made precise in the definition below (similar to what is defined in [18]).

On the other hand, each relation R in a cleaning graph has also associated a set of *manual data repairs*. Roughly speaking, these represent the actions that can be performed by the user over the instances of that relation in order to repair some quality problems, typically made apparent by one or more quality constraint labelling that relation or a relation "ahead" of R in the graph. Some actions found relevant for the manual repair of data are: (i) the deletion of a tuple; (ii) the insertion of a tuple; and (iii) the update of the values of an attribute (other actions could be added as needed, without requiring further complexity). For the convenience of the user, it might be helpful to filter the information available in R. For this reason, we have considered that data repair actions are defined over updatable views of R. It is of no particular importance for this paper how updatable views are described. These can range from SQL expressions to relational lenses [2].

Definition 2.1: *A* Manual Data Repair *m over a relation $R(A_1, ..., A_n)$ consists of a pair $\langle view(m), action(m)\rangle$, where $view(m)$ is an updatable view over R and $action(m)$ is one of the actions that can be performed over $view(m)$:*

$$action ::= \textbf{delete} \mid \textbf{insert} \mid \textbf{update}\ A_i$$

In the case where the action is **update** *A_i, we use $attribute(m)$ to refer to A_i. In the sequel, we use mdr as an abbreviation for manual data repair.*

Definition 2.2: *A* Data Cleaning Graph *$\mathcal{G}$ for a set of input relations R_I and a set of output relations R_O is a labelled directed acyclic graph $\langle G, \langle \mathcal{Q}, \mathcal{M}\rangle\rangle$ such that*

- *$nodes(G) \subseteq \mathcal{R} \cup \mathcal{T}$. We denote by $rels(G)$ and $trans(G)$ the set of nodes of G that are, respectively, relations and data transformations.*
- *$R_I \cup R_O \subseteq rels(G)$.*
- *$n \in R_I$ if and only if ${}^\bullet n = \emptyset$ and $n \in R_O$ if and only if $n^\bullet = \emptyset$ and ${}^\bullet n \neq \emptyset$.*
- *if $(n, m) \in edges(G)$, then either ($n \in \mathcal{R}$ and $m \in \mathcal{T}$) or ($n \in \mathcal{T}$ and $m \in \mathcal{R}$).*
- *if $T \in trans(G)$ then $\mathbb{I}_T$ is equal to $\{sch(R) : R \in {}^\bullet T\}$ and $\mathbb{O}_T$ is equal to $\{sch(R) : R \in T^\bullet\}$.*
- *if $R \in rels(G)$ then ${}^\bullet R$ has at most one element.*
- *$\mathcal{Q}$ is a function that assigns to every $R \in rels(G)$, a quality constraint over the set of relations behind R in G or in R_I, i.e., $\mathcal{Q}(R) \in \mathcal{L}(R_I \cup \{R' \in rels(G) : R' \leq_G R\})$ such that $\mathcal{Q}(R)$ is monotonic w.r.t. R, i.e., given a*

set of relation instances that satisfies $\mathcal{Q}(R)$, the removal of an arbitrary number of tuples form the instance of R does not affect the satisfaction of $\mathcal{Q}(R)$.

- *$\mathcal{M}$ is a function that assigns to every $R \in rels(G)$, a set of manual data repairs over R.*

The conditions imposed on data cleaning graphs (dcg, for short) ensure that (i) R_I and R_O are indeed input and output relations of the graph; (ii) relations are always connected through data transformations; (iii) the input and output schemas of a data transformation are those determined by their immediate predecessors and successors nodes in the graph; (iv) the instances of a relation in the graph result, at most, from one data transformation; (v) the quality constraints over a relation in the graph can only refer to relations that are either in R_I or behind that node in the graph and must be monotonic w.r.t. to the relation of the node. This last condition is necessary to ensure that quality constraints can be evaluated immediately after the data of the relation is produced, i.e., do not depend on data that will be produced later, by transformations ahead in the graph. This situation is particularly useful when, in the process of tuning a particular transformation, it is important to check immediately the quality constraint over the result without having to re-execute the whole graph.

The example sketched in Figure 3 illustrates the notion of dcg, mainly making use of SQL for expressing constraints and updatable views. The input relations of this dcg are Team and Pub and there is a single output relation, CleanPub. In the part of the graph that is shown, we can see that the node R3 is labelled with the quality constraint unique(aId, title). It is not difficult to conclude this is indeed a monotonic constraint over relations $\leq_G R$. Moreover, the function $\mathcal{M}$ of this dcg assigns to the node R3 a single manual data repair, Mdr3, that consists in the view V3 defined over R3 that returns only the tuples that are blamed for the violation of Qc3 (this is formally defined in the next section) and the action **delete**.

Although in this case the aim of the manual data repair associated to the node R3 is to allow the repair of the quality problems identified in that same relation, we found useful to support the independent specification of manual data repairs and quality constraints in dcgs. For instance, let us suppose that we want to clean a relation of sale records Order(itemId, itemDesc, itemPrice, phoneCode, phoneNum, street, city, state,zip) with the help of two tables — Code(phoneCode, city, state) and Zip(zip, city, state), populated with clean data (an example taken from [6]). This cleaning process can be achieved by the dcg presented in Figure 4. Data transformation T1 is responsible for applying a certain quality criteria to automatically correct those Order tuples whose values of phoneCode, city, and state do not match the values of the corresponding attributes in table Code. The transformation T1 updates the city or state to null whenever it is not able to correct them automatically accordingly to the phoneCode (e.g., the phone code in the order is too different from those stored in Code). The role of transformation T2 is similar but using the zip table. The quality problems that have to be analysed by the user can be easily expressed in terms of relations R1 and R2 while the manual data repairs have to be done directly in the input table Order. In fact, since data transformations T1 and T2 both try to correct the values of

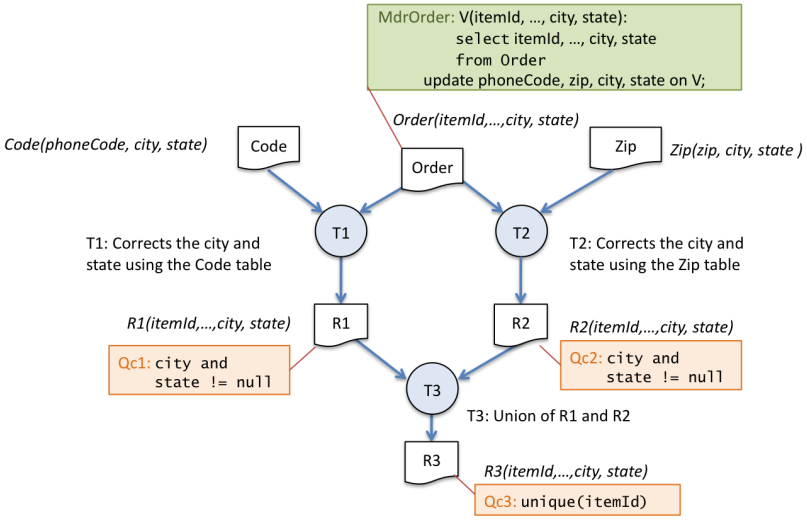


Figure 4: Excerpt of a data cleaning graph for cleaning the Order table.

the attributes city and state, the user feedback, when required, must be applied in the input relation.

While the definition of an mdr to solve the quality constraint of a relation has the advantage of supporting user feedback in a black-box manner (i.e., not requiring any knowledge about the data cleaning process), this is not always possible or even needed. In many situations, the person that will run the cleaning process is expected to know the underlying logic, perform manual repairs and even fine-tune the program data transformations. In these situations, the developer of the data cleaning process might define mdrs that, although require to know the high-level logic of the cleaning process, still facilitate the task of those that will run it on large amounts of dirty data.

2.2 Operational Semantics

Data cleaning graphs specify the quality criteria that the instances of each relation should meet. To help developers to identify where the problem is, the records responsible for the violation are identified. This requires to define a notion of blame assignment for quality constraints.

Definition 2.3: *Let ϕ be a quality constraint over a set of relations $R_1, ..., R_n$ that is assigned to relation R. Let r and $r_1, ..., r_n$ be instances of these relations s.t. $r, r_1, ..., r_n \not\models \phi$. The blame of the violation is assigned to the set blamed(ϕ), which is defined as the union of all subsets rp of r that satisfy:*

- $r \backslash rp, r_1, ..., r_n \models \phi$;
- *rp does not have a proper subset o s.t. $r \backslash o, r_1, ..., r_n \models \phi$.*

Intuitively, each subset rp of r that satisfies the two conditions above represents a way of "repairing" r through the removal of a set of tuples that, all together, cause the violation of ϕ (a particular case of data repairs as introduced

in [1]). Hence, all tuples in r that have this type of "incompatibility" share the blame for the violation of ϕ. For instance, suppose that R3 in Figure 3 has the tuples t =*(1, "Santos, A. L.", "Santos, A. L.", 2, "Managing...")* and t' =*(1, "Santos, A. L.", "Santos, A.", 3, "Managing...")*. These tuples are blamed for the violation of the quality constraint Qc3. Thus, they belong to the set given by $blamed(Qc3)$.

Notice that this form of blame assignment is only appropriate if constraints are monotonic in R and this is the reason why we limit constraints to be of this type. Blame assignment in an unconstrained setting is a much more complex problem, for which it is not clear that a good solution exist. For instance, if we consider constraints that are not satisfied by an empty relation, the notion of blamed tuples becomes unclear (there are no tuples to blame).

Note that assigning blame to a set of tuples in a relation for the violation of a constraint provides a semantics for constraints that is completly different from that adopted in RDBMS for integrity constraints. Constraints determine whether a database is consistent and, in RDBMSs, are used to avoid the insertion, removal or modification of tuples that would make the database inconsistent.

As mentioned before, data cleaning and transformation of a source of data tends to be the result of numerous iterations, some involving the tuning of data transformations and others involving manual data repairs. Even if the data cleaning graph developed for the problem was subject to a strict validation and verification process, it is normal that when it is executed over the real data, small changes in the data cleaning graph, confined to specific data transformations, are needed. Because we do not want to force the user to repeat the data repairs previously done that, in principle, are still valid, we define that the execution of a data cleaning graph takes as input not only the data that needs to be cleaned but also collections of instances of manual data repairs. These represent manual data repair actions enacted at some point in the past.

For convenience, we consider that instances of manual data repairs keep track of their type.

Definition 2.4: *Let m be a manual data repair. If $action(m)$ is* **delete** *or* **insert***, an $m-$instance ι is a pair $\langle m, tuple(\iota)\rangle$ where $tuple(\iota)$ is a $view(m)$-tuple. If $action(m)$ is* **update** *A, an $m-$*instance *ι is a triple $\langle m, tuple(\iota), value(\iota)\rangle$ where $tuple(\iota)$ is a $view(m)$-tuple, $value(\iota)$ is a value in $Dom(A)$.*

For instance, still referring to Figure 3, after analyzing the violation of the data quality constraint Qc3 and taking the title into account, the user could conclude that the author "Santos, A. L." does not correspond to the team author "Santos, A." and decide to delete the corresponding tuple from R3. This would generate the Mdr3-instance ⟨mdr3, (1, "Santos, A. L.", "Santos, A.", "Managing...")⟩.

The execution of a data cleaning graph is defined over a source of data (instances of the graph input relations) and what we call a *manual data repair state M* — a state capturing the instances of manual data repairs that have to be taken into account in the cleaning and transformation process. Because the order of actions in this context is obviously relevant, this state registers the order by which the instances of manual data repairs associated to each relation should be executed (what comes in first is handled first).

The execution of a data cleaning graph consists in the sequential execution of each data transformation in accordance with the partial order defined by the graph: if $T <_{\mathcal{G}} T'$, then T' is executed after T. The execution of a data transformation T produces an instance of the relation R in $T^{\bullet}$. This relation is then subject to the manual data repair instances in $M(R)$. Then, it is calculated the set of tuples in the resulting relation instance that are blamed for the violation of the quality constraint associated to R, $\mathcal{Q}(R)$. We say these tuples are "blamed".

Formally, the execution of a data cleaning graph can be defined as follows.

Definition 2.5: *Let $\mathcal{G} = \langle G, \langle \mathcal{Q}, \mathcal{M} \rangle \rangle$ be a data cleaning graph for a set $R_1, ..., R_n$ of input relations. Let $r_1, ..., r_n$ be instances of these relations and M be a* manual data repair state *for $\mathcal{G}$, i.e., a function that assigns to every relation $R \in rels(G)$, a list of instances of manual data repairs over R.*

The result of executing $\mathcal{G}$ over $r_1, ..., r_n$ and M is

$$\{\langle tuples(R), tuples^{bl}(R)\rangle : R \in rels(G)\}$$

calculated as follows:

1: ***for*** $i = 1$ *to* n ***do***
2: ***for each**** $\iota \in M(R_i)$ ***do***
3: $vr \leftarrow compute\_view(view(\iota), tuples(R_i))$
4: $apply\_mdr(\iota, vr)$
5: $tuples(R_i) \leftarrow propagate(vr)$
6: ***end for***
7: ***end for***
8: ***for*** $i = 1$ *to* n ***do***
9: $tuples^{bl}(R_i) \leftarrow blamed(tuples(r_i))$
10: ***end for***
11: ***for each**** $T \in trans(G)$ ***do***
12: **let** $\{R'_1, ..., R'_k\} = {}^{\bullet}T$
13: $tuples(T^{\bullet}) \leftarrow T(tuples(R'_1), ..., tuples(R'_k))$
14: ***for each**** $\iota \in M(T^{\bullet})$ ***do***
15: $vr \leftarrow compute\_view(view(\iota), tuples(T^{\bullet}))$
16: $apply\_mdr(\iota, vr)$
17: $tuples(T^{\bullet}) \leftarrow propagate(vr)$
18: ***end for***
19: $tuples^{bl}(T^{\bullet}) \leftarrow blamed(tuples(T^{\bullet}))$
20: ***end for***
21:
22: $apply\_mdr(mdrInstances, vr)$
23: ***for each**** $\iota \in mdrInstances$ ***do***
24: ***if*** $action(mdr(\iota)) =$ **delete** ***then***
25: $vr \leftarrow vr \setminus \{tuple(\iota)\}$
26: ***else if*** $action(mdr(\iota)) =$ **insert** ***then***
27: $vr \leftarrow r \cup \{tuple(\iota)\}$
28: ***else if*** $action(mdr(\iota)) =$ **update** ***then***
29: $newt \leftarrow tuple(\iota)$
30: $newt[attribute(action(mdr(\iota)))] \leftarrow value(\iota)$
31: $vr \leftarrow (vr \setminus \{tuple(\iota)\}) \cup \{newt\}$
32: ***end if***

33: ***end for***

**Assuming that the underlying iteration will traverse the set in ascending element order.*
***Assuming that the underlying iteration will traverse the list in proper sequence.*

The procedure $compute\_view(view, setOfTuples)$ encodes the application of the $view$ to the base table constituted by the $setOfTuples$ whereas $propagate(view)$ encodes the propagation of the updates applied to the tuples returned by $view$ to the base table.

Notice that the instances of manual repairs in each $M(R)$ are not limited to instances of the data manual repairs in $\mathcal{M}(R)$ and the algorithm presented above also does not check whether this is the case. This decision is justified by the fact that the role of manual data repairs, on the one hand, is to assist the user who is executing the process without forbiding him from applying mdrs he may find appropriate even though they are not specified. On the other hand, in the case of manual repairs put in place for end-users, these restrictions can be enforced by the system that establishes the interaction with the user.

Although this algorithm defines an operational semantics for data cleaning graphs, it must not be regarded as a proposal for the implementation of an engine that supports the execution of dcgs. The sole purpose of this algorithm is to formally define what is the result of executing a dcg over a source of data and a manual data repair state. The implementation of a system that supports the execution of dcgs is ongoing work.

3 Case Study

In order to evaluate our proposal, we developed in full depth the case study that was briefly introduced in Section 1. That is to say, we have developed and implemented a process to clean publication citation data retrieved from the web, aiming at determining with accuracy the research performance indicators for a research team. By using a data cleaning graph to model this process, we were able to define explicitly: *(i)* which data items deserve the attention of the user responsible for the execution of the process, and *(ii)* where user involvement in terms of manual correction is expected and in which form. Then, we performed a set of experiments to evaluate the benefits of involving the user in this data cleaning process. In these experiments, we focused on two different aspects: the data quality obtained at the end of the data cleaning process and the cost of the manual activities that have to be performed by the user.

The goal of our data cleaning process is to clean the Pub table (described in Figure 2) and produce a table containing only the publications authored by at least one team member, with duplicate entries for the same real world publication organized in clusters. In order to achieve this, we envisaged a process that handles the contents of the Pub records performing the following steps:

(i) extracting the author names independently of the publication they are associated to;

(ii) matching each of these author names against the names stored in the Team table, and try to find synonyms (i.e., approximate similar names);

(iii) building the list of co-authors for each author;

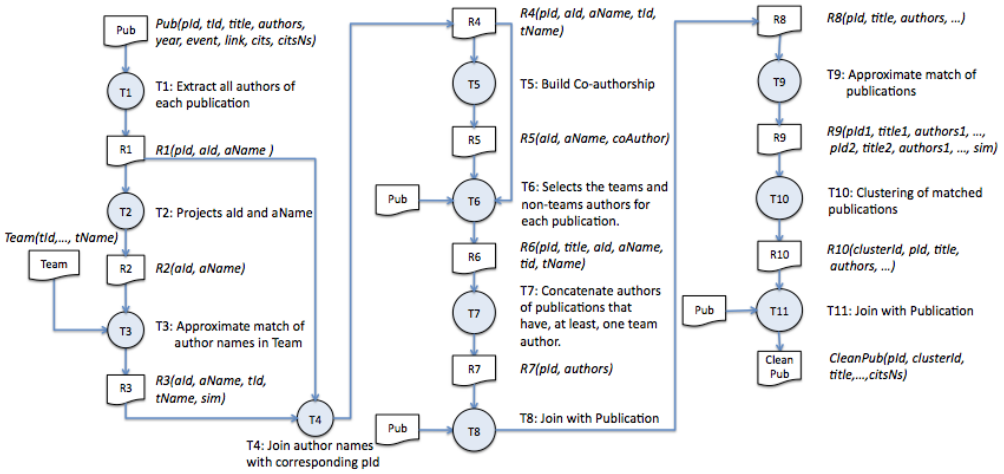


Figure 5: Data cleaning graph for the case study (for the sake of readability, the node Pub is depicted several times).

(iv) removing those publications that are not authored by any team member;

(v) detecting and clustering approximate duplicate publication records.

The calculation of co-authors in (iii) is a step towards the identification of homonyms of team members. The idea is to rely on user feedback to obtain accurate co-authorship information and, in this way, be able to filter out the publications authored by homonyms of team members.

Table 1: Quality constraints of the data cleaning graph

Node	Quality Constraint
Pub	Qc0: Pub.authors !contains ("others")
R3	Qc3: R3.sim $\geq$ 0.8
R6	Qc6: unique(pId, aId)
R8	Qc8: R8.authors !contains ("and")
R9	Qc9: R9.sim $\geq$ 0.8

Table 2: Manual data repairs of the data cleaning graph

Mdr	Node	User actions	View
Mdr0	Pub	delete, update author	Select title, authors From blamed(Qc0)
Mdr3	R3	update sim	Select aName, tName, sim From blamed(Qc3)
Mdr5	R5	delete	Select aName, coAuthor From R5
Mdr6	R6	delete	Select title, aName From blamed(Qc6)
Mdr8	R8	delete	Select title, authors,... From blamed(Qc8)
Mdr9	R9	update sim	Select title1, ..., title2, ..., sim From blamed(Qc9)

The data cleaning graph that models this process is presented in Figure 5 and in Tables 1 and 2. It presents slight differences with respect to the excerpt presented in Figure 3, because therein we made some simplifications for ease of presentation. The graph is composed by three main sub-flows of data transformations. The flow on the left (constituted by data transformations T1, T2, T3, and T4) aims at identifying synonyms of author names in the Team table. For those author names that definitely correspond to the members stored in this table, the corresponding tId and tName are associated. The flow at the center (constituted by data transformations T5, T6, T7, and T8) aims at constructing all the possible co-authorship relations that can be extracted from the publication records and obtaining confirmation for them from the user. The true co-authorship information is used to filter out publications that are not authored by any member of the team. Finally, the flow on the right (constituted by data transformations T9, T10, and T11) aims first at detecting Pub records that correspond to the same real publication. Then, it puts together in the same cluster all the duplicate Pub records.

Now, we detail each of the data transformations. Then, we motivate the user feedback and explain how we propose to integrate it into the graph of data transformations, through the use of quality constraints and manual data repairs.

The data transformation T1 applies to the input relation Pub(pId, title, authors, year, event, link, cits, citsNs) and extracts as many lines as authors a publication has, for each input record. It populates the output table R1(pId, aId, name), where the aId attribute is an identifier of each author name encountered, independently of the publication it was extracted from. Then, the data transformation T2 projects the attributes (aId, aName) and produces relation R2. Data transformation T3 applies to two tables: R2 and Team(tId,..., tName), where tName stores the standard name of the team member. T3 performs a lookup operation, which means that a string similarity function is applied. This function checks if values of pairs (Team.tName, R2.aName) have a similarity value above a pre-defined threshold. For each pair of these strings considered as similar, a tuple is inserted in the output relation R3(aId, aName, tId, tName, sim), where sim stores the similarity of each pair (Team.tName, R2.aName). If no matching string is found in Team.tName for a given value of R2.aName, then no output tuple is inserted in the output relation R3. Data transformation T4 joins the input table R3(aId,aName, tId, tName, sim) with the R1 table so that the publication identifiers are again associated with the corresponding author names. The output table is R4(pId, aId, aName, tId, tName).

Data transformation T5 takes as input table R4 and iterates over all its tuples for a given author name, collecting the names of the people that co-author publications. The output relation is R5(aId, aName, coAuthor). For example, this table stores five co-authors for "Carrico, L.", according to the records of the Pub table shown in Figure 2. In reality, two of these co-authors ({"Pereira, C."; "Gil, L."}) are not co-authors of the person named "Carriço, L." who belongs to the team. Data transformation T6 takes as input relations R5, R4 and Pub and associates the publication identifiers and title with the corresponding author names and team identifiers. The output table is R6(pid,title,aId,aName,tId,tName). If an author of a given publication is not considered a team member by the user-feedback provided via Mdr5 (as explained below), the corresponding team identifiers (tId, tName) have a null value in R6. For example, the fact that the team member "Carrico, L." is not a co-author of "Pereira, C." can be expressed via

an Mdr5 instance by deleting the corresponding tuple from table R5. Then, after executing T6, the tuple of R6 that corresponds to the publication with pId equal to 4 and author name Carrico, L. will have null values in the attributes tId and tName.

Data transformation T7 takes as input table R6 and concatenates into a single string the authors for each publication that has at least one team author. The output table is R7(pId,authors). Finally, data transformation T8 joins R7 with the initial Pub relation using the pid attribute in the join condition, such that the relation produced, R8, has a schema similar to the initial Pub relation.

Besides producing Pub records that concern only team members, the goal of the graph is also to put together Pub records that concern the same real world publication. This is required for computing the correct citation values for a given publication. To this end, the publication records must be compared, attribute by attribute, in order to identify entries that constitute approximate duplicates. For this purpose, the transformations T9 and T10 were included in the graph. These transformations match pairs of publications, and cluster the matched publications, respectively. Finally, data transformation T11 joins the R10 relation with the initial Pub relation using the pid attribute so that attributes link, CitsNs, cits are included in the schema of the final relation CleanPub.

The condition that an author of each publication can only match one team member is checked through the quality constraint Qc6 that is imposed after the user gives feedback about the co-authorship relationship (through the mdr associated to R5, i.e., Mdr5). Recall that data transformation T5 was introduced for gathering the co-authorship information about each author. The underlying idea is that the co-authorship information, after being validated by the user, can provide additional knowledge that is helpful for automatically deciding whether an author name in a publication refers to a team member.

Based on the matching name pairs produced by data transformations T3 and T4, and on the co-authorship relations produced by data transformation T5, the data transformation T6 is able to distinguish, among the set of authors for each publication, those who belong to the team from those who do not. The user feedback provided through the mdr associated to R6 (Mdr6) confirms whether the information automatically produced is true.

Other quality constraints were introduced in the graph to call the user's attention for antecipated data problems (see Table 1). Qc0 and Qc8 call the user attention for analyzing and correcting tuples that have the word "others" in its authors attribute value, and tuples that correspond to single-author publications, respectively. In particular, the quality constraint Qc0 guarantees that the list of authors is exhaustive (since the word "others" referring to the rest of the authors is not permitted) and the quality constraint Qc8 checks if the author attribute value does not contain the conjunction "and", which connects two or more authors names. Quality constraints Qc3 and Qc9 are imposed on the result of the matching operations encoded in T3 and T9, respectively. The matching criteria enclosed in these data transformations considers the existence of two threshold values. Pairs of records whose computed similarity is below the inferior threshold are considered as non-matches and discarded by the transformations. Pairs of records whose similarity is above the inferior threshold are considered as candidate matches and returned as a result of the data transformations. Those resulting records whose similarity value (stored in the sim attribute) is inferior to the superior threshold violate the corresponding quality constraints (Qc3 and

Qc9). These records do not have a sufficiently high value nor a sufficiently low value of the sim attribute, so the user must analyze them. Then, through the manual data repairs Mdr3 and Mdr9 (see Table 2), the user may decide whether the corresponding pairs of author names or publications are considered as matches or no matches, by modifying the sim attribute value accordingly (1 for matches, and 0 for no matches).

4 Experiments

The experiments we developed to evaluate our approach were performed with the support of the AJAX data cleaning prototype [12], over a dataset that is a subset of the database of the CIDS project [7]. These experiments required to implement two data cleaning programs:

- P_1 complying with the data transformation graph presented in Figure 5;
- P_2 complying with the data transformation graph presented in Figure 5 and capturing, as closest as possible with the means available, the quality constraints presented in Table 1.

Quality constraints in P_2 were encoded inside transformations, making use of exceptions as supported by AJAX. As a result, the tuples available for user inspection are not those blamed for the violation but those that originate a blamed tuple. Moreover, the tuples that raise exceptions are not available as input for the transformations ahead in the graph. However, for the evaluation purpose at hand, these differences were considered to be neglectable.

Moreover, we performed the following cleaning tasks:

T_1 : the manual cleaning of the Pub table;

T_2 : the execution of P_1 and the manual intervention of the user over the produced data in the output CleanPub table so that the tuples included there meet the data cleaning goal (all publications that are authored by at least one team member with duplicates organized in clusters);

T_3 : the execution of the P_2 and the manual intervention of the user over the produced data in the output CleanPub table guided by the rejected tuples in the different points of the program;

T_4 : the execution of the data cleaning program and, after receiving user feedback, the re-execution of parts of it — with the user involvement guided by the rejected tuples and the manual data repairs presented in Table 2.

Notice also that the decision in T_2 of limiting user feedback to corrections in the output table is because, otherwise, the results of our evaluation would be conditioned by the expertise of the user in choosing the most appropriate places to perform the data corrections.

4.1 Metrics used

The metrics used to evaluate the data quality of the CleanPub records produced are recall and precision defined as follows.

- *TD Recall* is given by the number of CleanPub tuples that are authored by the team (i.e., are authored by at least one team member) divided by the number of CleanPub tuples authored by the team that should have been produced.
- *TD Precision* is given by the number of CleanPub tuples that are authored by the team divided by the number of CleanPub tuples that were produced.
- *DD Recall* is given by the number of pairs of CleanPub tuples that were correctly identified as duplicates (i.e., the ones with the same value of the clusterId attribute and that correspond to the same real publication) divided by the total number of pairs of CleanPub tuples that should have been identified as duplicates.
- *DD Precision* is given by the number of pairs of CleanPub tuples that were correctly identified as duplicates divided by the number of pairs of CleanPub tuples that were identified as duplicates.

To evaluate the cost associated to the user feedback, we computed: (i) the number of characters the user needs to visualize in order to decide which data corrections need to be undertaken; (ii) the maximum number of characters that may need to be updated, when attribute values are modified; (iii) the maximum number of characters that may need to be deleted or inserted, when tuples are deleted or inserted; and (iv) the number of tuples that need to be updated, deleted or inserted. The number of characters is given by the product of the number of tuples by the sum of the sizes of each attribute.

4.2 Data cleaning of the CIDS database

We used an instance of the CIDS database containing 509 tuples in the Pub table and 24 tuples in the Team table. This instance was built so that it includes all the publication records returned by Google Scholar for five members of the team, chosen beforehand. These five team members are: "Antunes, P."; "Carriço, L."; "Lopes, A."; "Vasconcelos, V.", and "Santos, A. L.". First, we performed task T_1 and obtained the cleaned version of this instance by manually cleaning it. This process was performed by retrieving information from the member's home pages and the DBLP site. Then, the cleaned Pub table obtained was checked and eventually corrected by each team member. The manually cleaned publication table, that we named $\mathsf{CleanPub}_1$, was used as a reference for computing the quality of the data cleaned automatically and the impact of user feedback.

4.2.1 Quality of the data produced by the cleaning process

To compute the gain of data quality obtained when incorporating the user feedback, we performed the tasks T_2, T_3 and T_4. The resulting publication records obtained in each of these cases were stored in tables named, respectively, $\mathsf{CleanPub}_2$, $\mathsf{CleanPub}_3$, and $\mathsf{CleanPub}_4$. The recall and precision (i.e., both TD and DD) of the $\mathsf{CleanPub}_3$, and $\mathsf{CleanPub}_4$ tables were 100%. We recall that, in both cases, the manual corrections applied by the user are guided by rejected tuples. In the case of $\mathsf{CleanPub}_2$, 70% of TD recall, 78% of DD recall and 100% of precision were obtained. In fact, in Task T_2, the user only has access to the data produced at the end of the data cleaning process and so there is no

way of recovering the data tuples that were not properly handled by some data transformations.

Overall, these data quality values can be considered as good. However, as we will explain in Section 4.2.2, there is a trade-off between data quality and the cost of user feedback required.

Table 3: Precision and Recall results for the table $\mathsf{CleanPub}_4$.

mdr	TD Precision	TD Recall	DD Precision	DD Recall
none	0.83	0.70	0.98	0.76
Mdr0	0.83	0.70	0.98	0.76
Mdr3	0.85	0.80	0.98	0.91
Mdr5	1	0.92	0.98	0.91
Mdr6	1	0.92	0.98	0.91
Mdr8	1	1	0.93	0.93
Mdr9	1	1	1	1

In the case of task T_4, to analyze the effect of the different mdrs in the final result, we measured the values of precision and recall after applying each mdr. We considered that after the mdr instances were applied, the remaining of the data cleaning graph was re-executed and the precision and recall of $\mathsf{CleanPub}_4$ data was re-computed. The results obtained are summarized in Table 3. In this table, we notice that the precision and recall values greatly improved with the user's feedback via mdrs. The non-increasing values of DD precision when Mdr8 is applied are justified by the existence of pairs of tuples that correspond to the same single-author publication but whose similarity is inferior to 0.8. These pairs of tuples violated Qc8 and, because we use AJAX exception mechanism for "simulating" quality constraint violation, they were not delivered to transformation T9.

4.2.2 Cost of user feedback

The different cleaning tasks we have performed also allow us to evaluate, from a particular point of view, the cost associated to the user effort in providing feedback. The goal is to find out whether the approach we propose of incorporating the user feedback into the data cleaning graph (embodied by T_4) facilitates the work of user when compared to other approaches.

For this purpose, we measured the cost associated to the user actions performed in the four tasks mentioned above. The results that were obtained are presented in Table 4. The cost of data visualization, update, delete and insertion are approximate values.

In Table 4, we notice that the use of quality constraints and mdrs in Task T_4 greatly decreases the cost of data visualization with respect to the other tasks. Notice that this result is even true when comparing the cost of data visualization incurred in Task T_2, which only considers the data produced at the end of the data cleaning process. This result can be explained by the existence of quality constraints that were specified in such a way that only the set of tuples blamed by constraint violations are shown to the user. In other cases, the manual data repairs define judiciously the data the user needs to analyze in order to decide which action must be applied.

In what concerns the cost of the user feedback incurred in each task, we also observe that the use of mdrs also decreases substantially the number and cost

Table 4: Cost of user feedback

Cleaning Task	T_1	T_2	T_3	T_4
Cost of Data Visualization	200.000	137.000	115.000	32.000
# deleted tuples	164	56	56	134
Cost of delete	33.500	11.500	11.500	7.500
# updated tuples	121	2	32	21
Cost of update	2.600	40	800	150
# inserted tuples	0	0	68	0
Cost of insertion	0	0	14000	0

of user actions that must be applied to manually correct data. In comparison to Tasks T_1 and Task T_3, the results obtained by Task T_4 are significantly improved. Although in Task T_4 the user deletes a higher number of tuples than in Task T_3, the cost of delete in Task T_4 is lower than the corresponding cost in Task T_3 because the user has to analyse a smaller amount of data in order to apply each delete action. With respect to Task T_2, the obtained results are slightly better than Task T_4 because in Task T_2 the user actions are only applied over data produced at the end of the data cleaning process and, therefore, the rejected tuples are not analyzed, resulting in significantly worst recall values (70% of TD Recall and a 78% of DD Recall).

Overall, the results show that the use of the new primitives addressing the user feedback (T_4) may significantly improve a data cleaning process. We do believe that there are also significant gains in terms of time users need for deciding where and how to step in. Experiments for testing this hypothesis are planned for the near future.

5 Related work

First, we explain how commercial ETL and data cleaning tools in general handle the occurrence of errors. We also mention some research initiatives in the context of ETL and data cleaning that have proposed the integration of human intervention in a flow of data transformations. Then, we describe how the notion of user feedback has been recognized as important in related contexts, namely in Information Extraction. Finally, we summarize the work that has been developed around the concept of data repairs, since it is related to the definition of blamed tuples as we propose in this paper.

5.1 Error handling in ETL and data cleaning tools

In general, current commercial ETL and data cleaning technology supports the notion of a log file associated to the execution of a flow of data transformations. The developer can specify that the input records that are not handled by some pre-defined operators should be written into a log file. The contents of this file can be later analyzed by the user. However, no user feedback provided on the data stored in these files can be re-integrated in the flow of data transformations.

In some tools (e.g. SQL Server Integration Services), it is possible to partially overcome this limitation, by explicitly specifying an error output flow for some

data operators. Records inserted in this flow can be later analyzed by the user or considered as input of further data operators. The types of errors that may produce records to be inserted in the error output flow are pre-defined (data conversion, expression evaluation or lookup errors) and there is no way of specifying error conditions. Furthermore, once the errors are inserted into the error output flow, there is no mechanism available to re-integrate possible corrections performed by users into the data transformation flow. We provide some examples in what follows.

In relational engines, whenever an SQL query raises an exception, its execution is halted and the error cause is displayed. Exceptions occur when an integrity constraint imposed over a table is violated. In Oracle Warehouse Builder (OWB)[20], errors that occur during the execution of DML (Data Manipulation Language) operations (i.e., queries) are stored in an error table. The execution of these operations can continue despite the occurrence of errors. The error table contains Oracle error numbers, Oracle error message text, the rowid of the row in error (for update and delete), etc. The user can examine the errors occurred by querying this table. However, there is no mechanism to assist the user for correcting the tuples stored in this table.

In Talend [26], each data operator transformation is implemented in Java. Therefore, the Java exception mechanism is available and can be used to throw exceptions during the execution of an operator. An exception thrown during the execution of an operator can be caught in a parallel ETL flow by using the tLogCatcher operator. One of the tDie or tWarn operators can be inserted in the graph of data transformations after the operator that potentially throws an exception. They determine whether the flow ends or continues after the occurrence of the exception. The tLogRow operator can be used together with tLogCatcher to display the tuples that cause an exception to be thrown. Again, there is no support provided for incorporating manual corrections performed by the user into the ETL flow, after the occurrence and handling of exceptions.

Support for error handling in the context of data cleaning was investigated in the context of the AJAX data cleaning prototype [13] and ARKTOS [27] through the notion of, respectively, *exception* and *rejection*.

In ARKTOS, an activity is a logical abstraction that corresponds to a data transformation performed by code. The definition of an activity includes a set of input schemata and an output schema, as our notion of data transformation. An activity is also associated to a rejections schema that holds the input rows that do not pass a condition imposed by the activity semantics or whose values are not considered appropriate for the transformation. By default, the rejection schema is the same as the input schema. Rejected tuples may be object of further examination by the DW administrator or simply ignored. In the first case, they must be stored in an intermediate recordset. In the second case, they are simply discarded.

AJAX is a data cleaning framework that encloses five new operators for data cleaning as an extension to the relational algebra. A data cleaning program is modeled as a graph of data transformations, where each transformation is implemented by a data cleaning operator. The semantics of each operator includes the generation of exceptions that correspond to input tuples that cannot be automatically handled. Exceptions may be produced by the external functions called within each operator or may be generated whenever an integrity constraint imposed over the output table of each operator is violated.

Exceptions are stored in an exception output table that is produced by each operator. At any stage of the execution of a data cleaning program, users may inspect exceptions, as well as regular tuples, and their provenance in the graph of data transformations. Then, they are allowed to manually correct exceptions or the records that caused them to occur. The corrected tuples may then be re-integrated into the data flow graph.

The two notions, exceptions and rejections, are similar in the sense that they consist of input tuples that are not properly handled by a given data transformation. AJAX also considers as exceptions those input tuples of a data transformation that produce tuples violating an integrity constraint imposed on the output table of the transformation. Rejected tuples and exceptions are stored in a specific table whose schema is the same as the input schema of the transformation (in ARKTOS) or contains the key of the input tuples (in AJAX). The purpose of this information is to call the user's attention for data items that are not correctly handled in specific points of the graph of data transformations.

This solution is quite pragmatic and might be adequate for dealing with low-level problems that can occur in transformations that make use of partial functions such as divisions, data types conversions, etc. However, it is not a principled approach and does not provide the support we believe should be available at the modelling level of data cleaning processes. For instance, AJAX exceptions rely on relational technology to detect the occurrence of integrity constraint violations and, as a result, in many situations it is not possible to predict which are the tuples that will be identified as exceptions because it will depend on the order in which tuples of the input tables are processed (typically not under the control of the developer). The exact tuple that will be considered an exception looks, from the perspective of an external observer, a non deterministic choice.

5.2 User feedback

The incorporation of user feedback has shown to be useful in several automatic tasks. For example, automatic information extraction and integration programs aim at retrieving data from several data sources, integrating it and populating data structures. These programs often commit mistakes and produce data with errors. Therefore, the user feedback becomes useful to improve the results obtained. Chai et al [5] propose a solution to incorporate the end-user feedback into Information Extraction programs. An Information Extraction program is composed by a set of declarative rules defined in the *hlog* language. The developer writes some of these rules with the purpose of specifying the items of data the users can edit and the user interfaces that can be used. Analogously, we are proposing a way of specifying the exact points in the graph of data transformations where the user can provide feedback to improve the quality of the produced data. Moreover, we are limiting the amount of information the user can visualize and provide some guidance for the manual modification of data.

Other domains where the importance of user feedback has been recognized are dataspaces [16], data integration systems [19], schema matching [8], data cleaning [22], and entity resolution [24].

5.3 Data Repairs

The authors of the Conquer system [11], whose goal is to answer queries against inconsistent data, provide a definition of integrity constraint violation that is different from the one we provide in this paper. For them, dirty data is data that contains errors. Data may be cleaned and yet not satisfying integrity constraints. In this case, data is said to be inconsistent or to contain exceptions.

In [6], Cong and colleagues propose a framework for data cleaning that supports algorithms for finding repairs for a database and a statistical method to guarantee the accuracy of the repairs found. A database is considered dirty whenever data violates a set of conditional functional dependencies (CFD), as defined in [10] and [3]. A repair of a database D with respect to a set of CFDs is then defined as a database D' that satisfies the set of CFDs and is obtained from D by means of a set of repair operations. Furthermore, D' should be such that it minimally differs from D according to a certain cost function. In this context, repair operations consist in attribute value modifications. Tuple insertions are not considered since they do not lead to repairs when CFDs are considered.

As already mentioned in Section 2.2, the notion of blamed tuples introduced in this paper is based on this concept of database repair (considering that repair operations are limited to deletion of tuples). We consider as blamed for the violation of a data quality constraint associated to a relation of a database, those tuples in the relation instance that belong to some repair of the database.

In a recent demonstration paper, [28] proposes GDR, a system for guiding data repairing. The system explicitly involves the user in the process of checking the data repairs automatically produced by the algorithms introduced in [6]. In particular, the authors focused on ranking the repairs in such a way that the user effort spent in analyzing useless information is minimized. More concretely, after being produced, data repairs are grouped such that the repairs that suggest the same value for a given attribute come together, for instance. Then, a benefit score is computed and assigned to each group of repairs. In this paper, we aim at reaching the same objective: to minimize the user effort when providing feedback in a data cleaning process. However, in the current version of our research, we do not provide any method for clustering or ranking the tuples that violate quality constraints according to any particular criteria. This is object of future work. For the moment, we claim that by disclosing a limited set of records to the user, we are able to reduce the amount of data that he/she needs to analyze and eventually modified.

6 Conclusions

In this paper, we address the problem of integrating the user feedback in an automatic data cleaning process. The interaction of the user is required, because some data records cannot be handled by any possible automatic data transformation. We claim that the user knowledge is crucial to improve the quality of the data produced by a data cleaning process, in particular when coping with large amounts of input data.

We propose the notion of *data quality constraint* that may be associated to any of the intermediate relations produced by data transformations in a dcg. The purpose of these constraints is to call the attention of the user for the

records that violate them. The records blamed for the violation correspond to data items not handled automatically. We think that the participation of the user that is responsible for running a data cleaning program may be crucial to solve these situations and thus improving the quality of the resulting data. However, this user is not required to have a deep knowledge of the underlying graph of data transformations. For this reason, we proposed that dcgs also specify *manual data repairs*, that to some extent can be regarded as a kind of wizard-based form that limits the amount of data that can be visualized and modified. This way, we intend to decrease the burden of the user when he is called to intervene during the cleaning process. It encloses an updatable view that defines the data of the underlying relation the user may visualize, and the type of action that can be applied to each record returned by the view.

We have performed preliminary experiments with a real-world data set to demonstrate the advantages of our approach. In particular, we show the gain of data quality achieved when the user feedback is incorporated and we show that the overhead incurred by the user, when providing feedback guided by quality constraints and manual data repairs, is significantly inferior to the effort involved in cleaning rejected records in an ad-hoc manner.

Ongoing and future work. Currently, we are implementing a second data cleaning application for another data domain, using AJAX. With this second experiment, we intend to validate whether the experimental conclusions obtained with the CIDS database in what concerns the user feedback are confirmed. In addition, we plan to develop the same two data cleaning applications using a commercially available ETL or data cleaning tool (e.g., SQL Server Integration Services) and compare the results obtained in terms of data quality and user cost with our approach. This experience is important since the paradigm to cope with rejected tuples in these tools is completely different from what we are proposing, so we expect that the advantage brought by our approach is even more significant.

As far as future future work is concerned, we envisage four types of developments. First, we should include the possibility that data transformations may not be able to handle certain input data records. In this case, data transformations correspond to partial functions, which means they are not defined for certain input data values. This corresponds to defining pre-conditions on the input tuples of a data transformation that must be verified, which is the same as saying that there are quality constraints imposed on the output of the previous data transformation in the graph.

Second, the definition of updatable view that is used in the definition of manual data repair in this paper should be modified so that the join of base relations is possible. However, special care must be taken so that the view remains updatable in the sense that the updates can always be propagated to the base relations.

Third, we should consider that the set of manual data repairs instances may have contradictory user actions (e.g., distinct updates on the same tuple). Up to now, we resolve this problem by selecting the most recent mdr instances. We should take into consideration, for example, the different user expert degrees. We think that argumentation [23] can play an important role in deciding which mdr instances should be applied.

Fourth, the concept of data cleaning graph and corresponding operational

semantics must be adequately supported by a software platform. Up to now, we have been using the AJAX data cleaning framework to produce experimental evidence for our proposal. However, as already mentioned, it does not support exactly the notions of quality constraints (as well as blamed tuples) and manual data repairs. This new software platform should efficiently compute the set of blamed tuples for a given quality constraint violation, enable the automatic re-application of past user actions, and support the incremental execution of data transformations.

References

[1] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79, 1999.

[2] A. Bohannon, B. C. Pierce, and J. A. Vaughan. Relational lenses: a language for updatable views. In *PODS '06: Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 338–347, New York, NY, USA, 2006. ACM.

[3] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE*, pages 746–755, 2007.

[4] P. Carreira, H. Galhardas, A. Lopes, and J. Pereira. One-to-many data transformations through data mappers. *Data Knowl. Eng.*, 62(3):483–503, 2007.

[5] X. Chai, B.-Q. Vuong, A. Doan, and J. F. Naughton. Efficiently incorporating user feedback into information extraction and integration programs. In *SIGMOD Conference*, pages 87–100, 2009.

[6] G. Cong, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving data quality: Consistency and accuracy. In *VLDB*, pages 315–326, 2007.

[7] F. M. Couto, C. Pesquita, T. Grego, and P. Verissimo. Handling self-citations using google scholar. *International Journal of Scientometrics, Informetrics and Bibliometrics*, 13(1), 2009.

[8] A. Doan, P. Domingos, and A. Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *SIGMOD Conference*, pages 509–520, 2001.

[9] L. Egghe. Theory and practise of the g-index. *Scientometrics*, 69(1):131–152, 2006.

[10] W. Fan, F. Geerts, and X. Jia. Conditional dependencies: A principled approach to improving data quality. In *BNCOD*, pages 8–20, 2009.

[11] A. Fuxman, E. Fazli, and R. J. Miller. Conquer: Efficient management of inconsistent databases. In *SIGMOD Conference*, pages 155–166, 2005.

[12] H. Galhardas, D. Florescu, D. Shasha, and E. Simon. Ajax: An extensible data cleaning tool. In *SIGMOD Conference*, page 590, 2000.

[13] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C.-A. Saita. Declarative data cleaning: Language, model, and algorithms. In *VLDB*, pages 371–380, 2001.

[14] P. A. V. Hall and G. R. Dowling. Approximate string matching. *ACM Comput. Surv.*, 12(4):381–402, 1980.

[15] J. Hirsch. An index to quantify an individual's scientific research output. In *Proceedings of the National Academy of Sciences*, 2005.

[16] S. R. Jeffery, M. J. Franklin, and A. Y. Halevy. Pay-as-you-go user feedback for dataspace systems. In *SIGMOD Conference*, pages 847–860, 2008.

[17] I.-S. Kang, S.-H. Na, S. Lee, H. Jung, P. Kim, W.-K. Sung, and J.-H. Lee. On co-authorship for author disambiguation. *Inf. Process. Manage.*, 45(1):84–97, 2009.

[18] O. Maimon and L. Rokach, editors. *The Data Mining and Knowledge Discovery Handbook*. Springer, 2005.

[19] R. McCann, A. Doan, V. Varadarajan, A. Kramnik, and C. Zhai. Building data integration systems: A mass collaboration approach. In *WebDB*, pages 25–30, 2003.

[20] Oracle. Oracle warehouse builder documentation. http://www.oracle.com/pls/db112/portal.portal_db?selected=6.

[21] E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

[22] V. Raman and J. M. Hellerstein. Potter's wheel: An interactive data cleaning system. In *VLDB*, pages 381–390, 2001.

[23] E. Santos, J. Martins, and H. Galhardas. An argumentation-based approach to database repair. In *ECAI'10: Proceedings of the 19th European Conference on Artificial Intelligence*, 2010.

[24] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *KDD*, pages 269–278, 2002.

[25] A. Simitsis, P. Vassiliadis, M. Terrovitis, and S. Skiadopoulos. Graph-based modeling of etl activities with multi-level transformations and updates. In *DaWaK*, pages 43–52, 2005.

[26] Talend. Talend open studio user and reference guides. http://www.talend.com/resources/documentation.php.

[27] P. Vassiliadis, A. Simitsis, P. Georgantas, M. Terrovitis, and S. Skiadopoulos. A generic and customizable framework for the design of etl scenarios. *Inf. Syst.*, 30(7):492–525, 2005.

[28] M. Yakout, A. K. Elmagarmid, J. Neville, and M. Ouzzani. Gdr: a system for guided data repair. In *SIGMOD Conference*, pages 1223–1226, 2010.