Semandaq: A Data Quality System Based on Conditional Functional Dependencies

Wenfei Fan[1,2,*] Floris Geerts[1,†] Xibei Jia[1,†]
[1]University of Edinburgh [2]Bell Laboratories
{wenfei@inf., fgeerts@inf., xibei.jia@}ed.ac.uk

ABSTRACT

We present SEMANDAQ, a prototype system for improving the quality of relational data. Based on the recently proposed *conditional functional dependencies* (CFDs), it detects and repairs errors and inconsistencies that emerge as violations of these constraints. We demonstrate the following functionalities supported by SEMANDAQ: (a) an interface for specifying CFDs; (b) a visual tool for automated detection of CFD violations in relational data, leveraging efficient SQL-based techniques; (c) extensive visual data exploration capabilities that provide the user with various measures of the quality of the data; (d) repair (cleaning) functionality without excess human interaction, built upon CFD-based cleaning algorithms; we show how SEMANDAQ allows for a natural exploration of the quality of the obtained repairs. SEMANDAQ is a promising tool that provides easy access and user-friendly data quality facilities for any relational database system.

1. INTRODUCTION

The prevalent use of information systems has made data one of the most valuable assets in most organizations. Nevertheless, the value of data highly depends on its quality. Errors and inconsistencies in the data dramatically reduce the value of data, making it worthless, or even harmful. A study conducted by Gartner in 2005 [10] forecasts that more than 50 percent of data warehouse projects will have limited success, or will be outright failures, as a result of the lack of attention to data quality issues. Indeed, useful results are unlikely to be obtained from unreliable source data, a general principle best known as "Garbage In, Garbage Out". Thus, to reduce the resources wasted on dirty data and avoid the potential disastrous consequences caused by errors, it is desirable to clean the data before using it in data warehousing, data mining, and business intelligence systems. It is estimated that data cleaning, a labor-intensive and complex process, accounts for 30%-80% of the development time in a data warehousing project [12]. These highlight the increasing need for data quality tools to automatically detect and effectively remove inconsistencies and errors in data.

[*]Adjunct professor, Harbin Institute of Technologies. Supported in part by EPSRC GR/S63205/01, GR/T27433/01, EP/E029213, and BBSRC BB/D006473/1.
[†]Supported in part by EPSRC EP/E029213.

PVLDB '08, August 23-28, 2008, Auckland, New Zealand
Copyright 2008 VLDB Endowment, ACM 978-1-60558-306-8/08/08

Existing data cleaning systems are dominated by transformation-based approaches, in which cleaning actions have to be explicitly specified by users. Data cleaning systems reported in the research literature include AJAX [7] and Potter's Wheel [11]. AJAX provides users with five transformation operations and a declarative language to specify data cleaning programs. Potter's Wheel leverages a spreadsheet-like interface to allow users to refine the cleaning process by interactively specifying transformations that automatically trigger error detections in the background. Most commercial data quality systems focus on data profiling, which reveals data quality issues by analyzing the data, and record matching, which removes duplicate entities in the dirty data. Their data cleaning capabilities are often skewed toward very specific types of data (e.g., addresses, phone numbers). They are based on ad-hoc techniques such as manually specified IF THEN rules where the actions in THEN clauses are usually a set of transformation operations.

A more principled approach to cleaning is based on constraints. Two models have been studied. One is *consistent query answering* (*e.g.,* [1, 6]), which can be seen as *virtual data cleaning*, without editing the data. The other is constraint-based repairing, which fixes constraint violations by *physically editing the data* (*e.g.,* [2, 5, 9]), a process that US national statistical agencies have been practicing for decades [4]. However, constraints considered in these models are mostly traditional dependencies, such as functional, inclusion and full dependencies, which were designed for schema design rather than data cleaning. Partly due to this, no commercial data-cleaning systems have emerged from these two models yet.

Following the constraint-repairing approach, we are developing SEMANDAQ (Semantic Data Quality), a data quality tool based on the concept of conditional functional dependencies (CFDs) recently introduced [3]. CFDs are an extension of functional dependencies (FDs) developed for data cleaning. They enforce bindings of semantically related data values, and are capable of capturing more errors and inconsistencies commonly found in real-life data than traditional dependencies can catch. SEMANDAQ allows users to specify a set of CFDs to characterize the semantics of data. Subsequently, errors and inconsistencies in the data are captured as *violations* of the CFDs. These violations are efficiently detected by SQL queries that are automatically generated from the given CFDs. The quality of data is then estimated based on the identified violations. Furthermore, SEMANDAQ uses CFDs to *repair* the violations and hence restore the consistency of the data; it presents the candidate repairs to users for inspections. Another functionality supported by SEMANDAQ is *incremental repairing*: the repaired data is monitored by the same set of CFDs, and any updates to the repaired data are incrementally detected and repaired by the monitoring process.

We believe that SEMANDAQ is a promising data quality tool that combines data exploration and data cleaning in a natural way. On

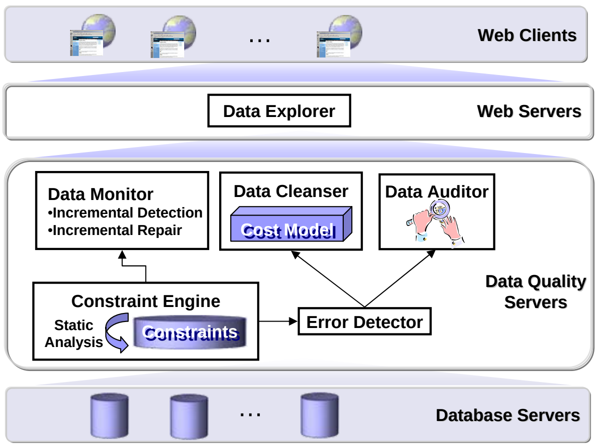


Figure 1: The SEMANDAQ Architecture

one hand, the functionality of SEMANDAQ helps the user quickly discover quality issues of the data and their solutions. On the other hand, the user's expertise is readily incorporated into SEMANDAQ's data cleaning process by efficient visual data exploration. This demonstration will show that SEMANDAQ's support for this two-way interactive process is extremely valuable for tackling data quality problems.

2. THE SEMANDAQ SYSTEM

The architecture of SEMANDAQ is depicted in Fig. 1. Below we briefly describe the major components of the system. We illustrate the CFD formalism, algorithms and their performance by examples in Section 3, and refer the reader to [3, 8] for details.

SEMANDAQ consists of four layers and six major components. Relational databases holding the data to be cleaned form the lowest layer of the system. The major functionalities are provided by the second layer consisting of a set of data quality servers. A data quality server is built upon of one or more of the following components: *constraint engine*, *error detector*, *data auditor*, *data cleanser* and *data monitor*. Each of these components is implemented as an Enterprise Java Bean (EJB). This allows the different components to run independently in a distributed way (possibly in different locations). Security and transactions are supported by the EJB container. Users can access these functionalities easily by means of the *data explorer*, which constitutes the third layer, and is built in a web container. Finally, the fourth layer allows users to access SEMANDAQ *from any computer with a standard web browser* by using Asynchronous JavaScript and XML technology.

Constraint Engine. The core of SEMANDAQ is the constraint engine, which manages the CFDs used to specify the consistency of the data. These constraints may either be explicitly specified by users or automatically discovered from reference data. Since CFDs allow for a relational representation [3], the constraint engine maximally leverages the use of indices and other optimizations provided by DBMS in the storage and manipulation of CFDs. Further, the constraint engine implements static analysis techniques developed in [3], that check for the consistency of CFDs. In this way, users are informed whether the specified set of CFDs "makes sense".

Error Detector. An important step in assessing the quality of data is to efficiently identify errors and inconsistencies. In SEMANDAQ, this is achieved by detecting violations of CFDs in the underlying relational databases. It hereby relies on efficient SQL-based detection techniques developed in [3]. Specifically, the SQL-queries identify two kinds of violations: (1) *single-tuple violations*, *i.e.,* a tuple that conflicts with a CFD all by itself; and (2) *multi-tuple violations*, *i.e.,* tuples that jointly conflict with a CFD in a similar way to that of functional dependencies.

Upon the completion of a detection process, the error detector assigns to each tuple t in the database the *number of violations incurred by* t, denoted by $\mathsf{vio}(t)$. Initially, $\mathsf{vio}(t) = 0$; it is incremented by 1 for each CFD for which t is a single-tuple violation; and is incremented for each CFD by the cardinality of the set of tuples that jointly (with t) violate that CFD. Moreover, the error detector records additional information that is useful for improving the quality of data (*e.g.,* which CFDs are violated by which tuple).

Data Auditor. This component provides a summarized report of the inconsistencies detected by the error detector. Specifically, $\mathsf{vio}(t)$ is enriched with statistical information *w.r.t.* the occurrences of violations in the data, at both the tuple and the attribute level.

Data Cleanser. To deal with inconsistent data, SEMANDAQ provides the user with an automatic repair functionality. The data cleanser passes the violation information provided by the error detector to the CFD-based repair algorithm developed in [8]. More specifically, a candidate repair is obtained from the original data using *attribute value modifications* on the violations. Moreover, the repair algorithm aims to find a repair that "minimally differs" from the original data (in terms of some cost function). In view of the intractability of this optimization problem [2, 8], the repair algorithm is heuristic in nature. It is reported in [8] that the repair algorithm generally provides candidate repairs of high quality.

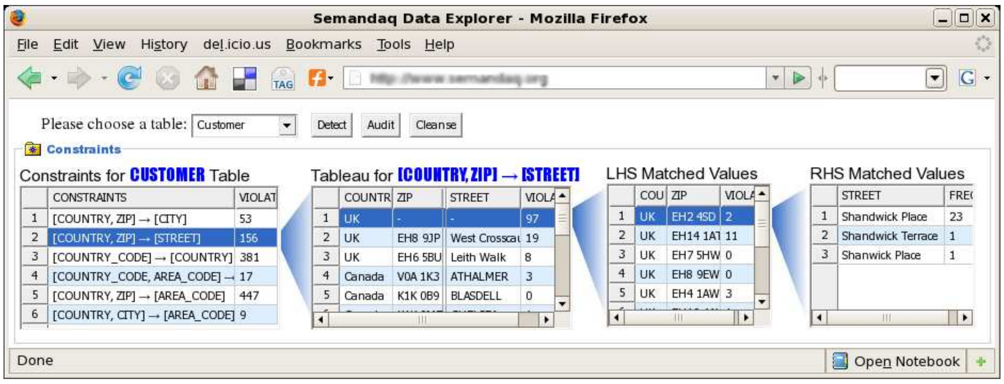
Data Monitor. Real world data changes with time and it is important to prevent the degradation of the quality when the data is updated. In SEMANDAQ, the data monitor responds to updates on the data by (1) invoking an incremental detection module (analogous to the error detector) using the incremental SQL-based detection techniques developed in [3] if the database has not been cleansed; or (2) invoking an incremental repair module (analogous to the data cleanser) using the incremental CFD-based repair algorithm developed in [8], otherwise.

Data Explorer. The final component of SEMANDAQ, namely, the data explorer, provides a graphical interface for the functionalities of the data quality servers. In the data explorer, one can specify CFDs, navigate the data, check the detection result, view the auditing information, review the cleansing result and monitor modifications on the data. The data explorer is important since it brings the human intelligence into the automated cleaning process and helps users understand the revealed data quality issues. Separated from the data quality server, the data explorer is designed as a rich Internet application (RIA) which provides a web-based interface. Thus, SEMANDAQ can be used on any client machine with a web browser without knowing where the data quality servers are running. It also enables the software as service mode for SEMANDAQ.

3. DEMONSTRATION OVERVIEW

In this section we describe various aspects of SEMANDAQ in more detail and explain the aims of our demonstration. More specifically, we show (a) how users define constraints in the form of CFDs with the aid of the data explorer; (b) how SEMANDAQ helps users browse the errors for large quantities of data; (c) how the quality of data is summarized by combining different metrics at different granularities; and finally (d) how to review and debug the data cleansing result.

Specifying Constraints. The first step in using SEMANDAQ is to connect the system to a database. Users simply need to provide a username, password, a JDBC url and the corresponding JDBC driver if it is not already included in the system. The database schema containing all the relations and attributes will be automatically dis-


Semandaq Data Explorer - Mozilla Firefox

Please choose a table: Customer

Constraints

Constraints for **CUSTOMER** Table

	CONSTRAINTS	VIOLAT
1	[COUNTRY, ZIP] → [CITY]	53
2	[COUNTRY, ZIP] → [STREET]	156
3	[COUNTRY_CODE] → [COUNTRY]	381
4	[COUNTRY_CODE, AREA_CODE] →	17
5	[COUNTRY, ZIP] → [AREA_CODE]	447
6	[COUNTRY, CITY] → [AREA_CODE]	9

Tableau for **[COUNTRY, ZIP] → [STREET]**

	COUNTR	ZIP	STREET	VIOLA
1	UK	-	-	97
2	UK	EH8 9JP	West Crosscau	19
3	UK	EH6 5BU	Leith Walk	8
4	Canada	V0A 1K3	ATHALMER	3
5	Canada	K1K 0B9	BLASDELL	0

LHS Matched Values

	COU	ZIP	VIOLA
1	UK	EH2 4SD	2
2	UK	EH14 1AT	11
3	UK	EH7 5HW	0
4	UK	EH8 9EW	0
5	UK	EH4 1AW	3

RHS Matched Values

	STREET	FREQ
1	Shandwick Place	23
2	Shandwick Terrace	1
3	Shanwick Place	1

Done

Figure 2: Data Exploration using CFDs

covered and displayed to the user. Once the schema information is available to the user, CFDs can be specified using the data explorer.

We first recall what CFDs are and then explain how they are specified by the user using the data explorer. We refer to [3] for more details concerning the CFD-formalism. Assume that a company maintains a relation of customer records: customer(NAME, CNT, CITY, ZIP, STR, CC, AC), where each customer tuple contains a name NAME, address (country CNT, city CITY, postal code ZIP, street STR) and country and area code (CC and AC, respectively) of a customer. Figure 3 shows an example of a customer relation. An example of a traditional functional dependency (FD) on a customer relation is f_1: [CNT, ZIP] $\rightarrow$ [CITY]. It requires that customer records with the same country and zip code also have the same city name. Traditional FDs are to hold on *all* tuples in the customer relation. In contrast, the following constraint is supposed to hold only when the country is UK. That is, for customers in the UK, ZIP determines STR:

$$\phi_2 : [\mathsf{CNT} = \texttt{UK}, \mathsf{ZIP} = \_\,] \rightarrow [\mathsf{STR} = \_\,]$$

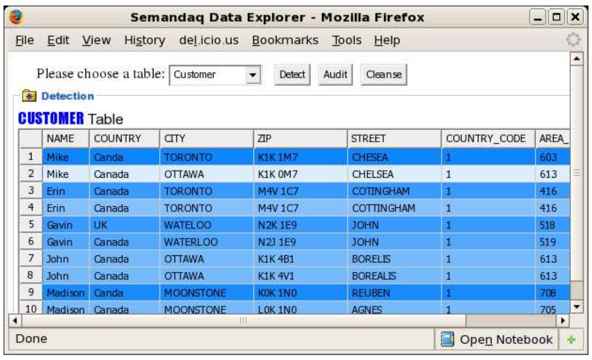
In other words, ϕ_2 is an FD that is to hold on the subset of tuples that satisfies the pattern "CNT = UK", rather than on the entire customer relation (the "_" symbol stands for a "don't care" value). It is generally *not* considered an FD in the standard definition since ϕ_2 includes a *pattern* with *data values* in its specification. We call such a dependency a *conditional functional dependency* (CFD) since the dependency holds provided that the condition stated by the pattern is satisfied.

Another example of a CFD relates to the FD f_3: [CC] $\rightarrow$ [CNT], which states that the country code determines the country. It is often the case that exact relationship between attribute values is known, *e.g.,* tuples with country code 44 should always have country name UK. Such constraints cannot be modeled by FDs, they can however be enforced by the following CFD: ϕ_4: [CC $=$ 44] $\rightarrow$ [CNT $=$ UK]. Observe that FDs f_1 and f_3 can be regarded as CFDs as well. Indeed, ϕ_1: [CNT $= \_$, ZIP $= \_$] $\rightarrow$ [CITY $= \_$] and ϕ_3: [CC $= \_$] $\rightarrow$ [CNT $= \_$] correspond to f_1 and f_3, respectively.

In short, a CFD is determined by (1) a standard FD $X \rightarrow Y$ embedded in it; (2) a left-hand side (LHS) pattern tuple $t_p[X]$ consisting of values and "_"; and (3) a right-hand side (RHS) pattern tuple $t_p[Y]$. The FD embedded is to hold on all tuples that satisfy the LHS-pattern, and moreover, these tuples should also satisfy the RHS-pattern tuple.

In the demonstration we will show how the user can specify CFDs using the data explorer. That is,

- the user first selects a relation (table) and creates a standard FD by dragging and dropping attributes;
- then, the user specifies the LHS and RHS patterns of the selected FD to make it a CFD. If the user specifies the pattern tuples manually, the system will provide auto-completion by querying the data instance. In this way, the user is provided with possible values for the pattern tuples. Alternatively, the user can import the pattern tuples from an external source.


Semandaq Data Explorer - Mozilla Firefox

Please choose a table: Customer

Detection

CUSTOMER Table

	NAME	COUNTRY	CITY	ZIP	STREET	COUNTRY_CODE	AREA_
1	Mike	Canda	TORONTO	K1K 1M7	CHESEA	1	603
2	Mike	Canada	OTTAWA	K1K 0M7	CHELSEA	1	613
3	Erin	Canada	TORONTO	M4V 1C7	COTINGHAM	1	416
4	Erin	Canada	TORONTO	M4V 1C7	COTTINGHAM	1	416
5	Gavin	UK	WATELOO	N2K 1E9	JOHN	1	518
6	Gavin	Canada	WATERLOO	N2J 1E9	JOHN	1	519
7	John	Canada	OTTAWA	K1K 4B1	BORELIS	1	613
8	John	Canada	OTTAWA	K1K 4V1	BOREALIS	1	613
9	Madison	Canda	MOONSTONE	K0K 1N0	REUBEN	1	708
10	Madison	Canada	MOONSTONE	L0K 1N0	AGNES	1	705

Done

Figure 3: Error Detection and Data Quality Map

In contrast to their classical FD counterparts, CFDs are not necessarily satisfiable [3]. In case that the CFDs specified by the users are not satisfiable (or don't make sense), they need to be checked before using them in a data quality setting. Therefore, the demonstration also shows

- how the user is prompted about the (un-)satisfiability of CFDs that are specified by the user.

Data exploration. The tight coupling of data and constraints in CFDs allows for an interactive exploration process between data and constraints. In the demonstration we will show the following.

- How the user can explore the data by means of CFDs. Indeed, consider Fig. 2. When the user selects an FD embedded in a CFD (*e.g.,* [CNT, ZIP] $\rightarrow$ [STR] in the left table of Fig. 2), its corresponding pattern tuples (second table in Fig. 2) will be displayed in a table. The user can then further select a specific pattern tuple t_p in this table (*e.g.,* tuple [UK, _, _] in Fig. 2). The data explorer will show all the distinct tuples t in the data which match the LHS pattern of t_p (third table in Fig. 2). A further selection of one of these LHS matched tuples (*e.g.,* tuple [UK, EH2 4SD]) then displays the distinct RHS values for the corresponding tuples in the data. In the fourth table of Fig. 2, three different RHS values are displayed. Finally, a selection of one of these RHS-values would provide the user with a list of the corresponding tuples in the data (not shown). It is worth noting that in each of the above steps, the number of violating tuples are given to the user to guide the navigation process. In Fig. 2, the number of violations appears in the violation-attribute.
- How the user can explore the CFDs by means of the data. Here, the user selects a tuple in the data and is provided with all CFDs and pattern tuples relevant to that tuple. In this way,

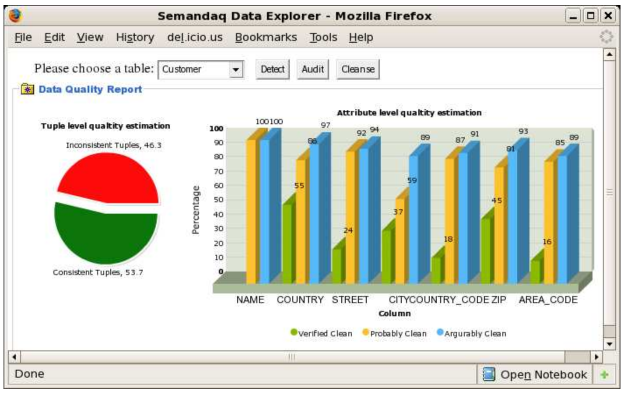


Figure 4: Data Quality Report

the user can identify the reasons why the tuple is regarded as a violation. Moreover, it also provides the users with necessary information to manually correct the data if he/she wishes to do so.

Data quality map. The error detection component provides valuable information regarding the violations of CFDs in the data. In the demonstration we will show:

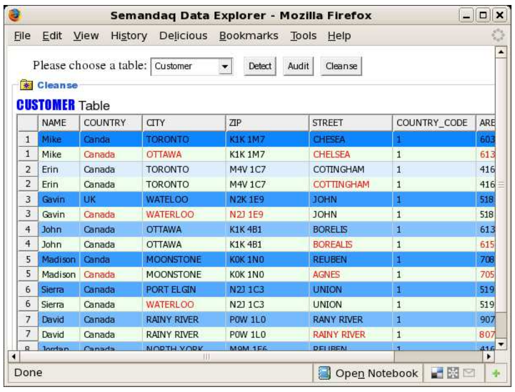
- how the result of the error detection component is visualized. In general, different colors are used to indicate the different numbers of violations for a tuple or a value. As an example, Figure 3 shows a tuple-level data quality map. The darker the color of a tuple is, the greater $\mathsf{vio}(t)$ is, and thus the more "dirty" the tuple is.

Data quality report. High level data quality information is provided by the auditor component. In the demonstration we show:

- How the level of dirtiness of the data is summarized to the user both at the table-tuple and the attribute-value level. In particular, the data auditor computes various data quality metrics which categorize each tuple t as "*verified clean*", meaning that t does not violate any CFD and moreover there exists a CFD with a constant in its RHS that applies to t (*i.e.,* the values in t are verified by at least one CFD); "*probably clean*", meaning that t does not violate any CFD; and finally "*arguably clean*", meaning that t is either probably clean or t is involved in a multi-tuple violation but the bulk of the joint violating tuples agree with t (*i.e.,* there is substantial evidence to regard t as being clean). A similar categorization exists at the attribute-value level. The bar chart in Fig. 4 shows the percentage of verified clean, probably clean and arguably clean values for each attribute in the customer relation.
- How simple data quality measures are reported; *e.g.,* Figure 4 shows a pie chart indicating the number of violations in the data. Various other measures can be reported in this way.
- How the violations are distributed among the data. The auditor computes various statistical measures (max,min, avg,...) and also reports statistics regarding multi-tuple violations. The user can choose to retrieve this information at different levels of details of the data.

Data cleansing review. In the final part of the demonstration we illustrate the interactive nature of the data explorer in combination with the data cleanser. More specifically, we show the following.

- How the candidate repair (cleansed data) compares to the original data. This is illustrated in Fig. 5 where the modified values are highlighted in red color. When the user selects a modified value, a pop-up window will show a list of alternative modifications. These alternatives are ranked according to the cost model used in the underlying repair algorithms in the data cleanser [8].


Semandaq Data Explorer - Mozilla Firefox
File Edit View History Delicious Bookmarks Tools Help
Please choose a table: Customer
Detect
Audit
Cleanse
Cleanse
Done
Open Notebook

CUSTOMER Table

	NAME	COUNTRY	CITY	ZIP	STREET	COUNTRY_CODE	ARE
1	Mike	Canda	TORONTO	K1K 1M7	CHESEA	1	603
1	Mike	Canada	OTTAWA	K1K 1M7	CHELSEA	1	613
2	Erin	Canada	TORONTO	M4V 1C7	COTINGHAM	1	416
2	Erin	Canada	TORONTO	M4V 1C7	COTTINGHAM	1	416
3	Gavin	UK	WATELOO	N2K 1E9	JOHN	1	518
3	Gavin	Canada	WATERLOO	N2J 1E9	JOHN	1	518
4	John	Canada	OTTAWA	K1K 4B1	BORELIS	1	613
4	John	Canada	OTTAWA	K1K 4B1	BOREALIS	1	615
5	Madison	Canda	MOONSTONE	K0K 1N0	REUBEN	1	708
5	Madison	Canada	MOONSTONE	K0K 1N0	AGNES	1	705
6	Sierra	Canada	PORT ELGIN	N2J 1C3	UNION	1	519
6	Sierra	Canada	WATERLOO	N2J 1C3	UNION	1	519
7	David	Canada	RAINY RIVER	P0W 1L0	RANY RIVER	1	907
7	David	Canada	RAINY RIVER	P0W 1L0	RAINY RIVER	1	807
8	Jordan	Canada	NORTH YORK	M9M 1E6	REUBEN	1	416

Figure 5: Data Cleansing Review

- How the user can review these modifications and change them.
- How changes to the suggested repair trigger a background incremental detection and how the effect of these changes to other tuples is visualized to the user by showing the conflicting tuples with this new value.
- How, for a large data set, this reviewing process could be combined with data exploring to concentrate only on interested part of the data.

In summary, the demonstration exhibits the strength of the CFD-formalism and automatic functionalities of SEMANDAQ (error detection, data cleansing, data auditing), and illustrates the savings of human effort in the data quality process. At the same time, the demonstration shows that SEMANDAQ provides the user with a better understanding of the quality of the data, assisting the user to improve the quality of the data in an interactive way.

4. REFERENCES

[1] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. *TPLP*, 3(4-5):393–424, 2003.

[2] P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*, 2005.

[3] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. *TODS*, 33(1), 2008. to appear.

[4] I. Fellegi and D. Holt. A systematic approach to automatic edit and imputation. *J. of the American Statistical Association*, 71(353):17–35, 1976.

[5] E. Franconi, A. L. Palma, N. Leone, S. Perri, and F. Scarcello. Census data repair: a challenging application of disjunctive logic programming. In *LPAR*, pages 561–578, 2001.

[6] A. Fuxman, E. Fazli, and R. J. Mille. Conquer: Efficient management of inconsistent databases. In *SIGMOD*, 2005.

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

[8] C. Gao, W. Fan, F. Geerts, X. Jia, and S. Ma. Improving data quality: Consistency and accuracy. In *VLDB*, 2007.

[9] A. Lopatenko and L. Bertossi. Complexity of consistent query answering in databases under cardinality-based and incremental repair semantics. In *ICDT*, 2007.

[10] Press release, Gartner, Inc., February 3, 2005, quoting Bill Hostmann, Research Director, presenting at Gartner Business Intelligence Summit in London, UK. *http://www.gartner.com/press_releases/asset_119071_11.html.*

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

[12] C. C. Shilakes and J. Tylman. Enterprise information portals. Technical report, Merrill Lynch, Inc., New York, NY, Nov. 1998.