NADEEF: A Generalized Data Cleaning System

Amr Ebaid[1,2*] Ahmed Elmagarmid[1] Ihab F. Ilyas[1] Mourad Ouzzani[1]
Jorge-Arnulfo Quiane-Ruiz[1] Nan Tang[1] Si Yin[1]

[1]Qatar Computing Research Institute (QCRI) [2]Purdue University

{aebaid, aelmagarmid, ikaldas, mouzzani, jquianeruiz, ntang, siyin}@qf.org.qa

ABSTRACT

We present NADEEF, an extensible, generic and easy-to-deploy data cleaning system. NADEEF distinguishes between a *programming interface* and a *core* to achieve generality and extensibility. The programming interface allows users to specify data quality rules by writing code that implements predefined classes. These classes uniformly define *what* is wrong with the data and (possibly) *how* to fix it. We will demonstrate the following features provided by NADEEF. (1) *Heterogeneity:* The programming interface can be used to express many types of data quality rules beyond the well known CFDs (FDs), MDs and ETL rules. (2) *Interdependency:* The core algorithms can interleave multiple types of rules to detect and repair data errors. (3) *Deployment and extensibility:* Users can easily customize NADEEF by defining new types of rules, or by extending the core. (4) *Metadata management and data custodians:* We show a live data quality dashboard to effectively involve users in the data cleaning process.

1. INTRODUCTION

Real-world data is dirty: more than 25% of critical data in the world's top companies is flawed [8]. Not surprisingly, dirty data inflicts a daunting cost: according to a recent study from Experian QAS Inc., poor customer data cost British businesses £8 billion loss of revenue in 2011 [3]. Despite the need of high quality data, there is no end-to-end off-the-shelf solution to (semi-)automate error detection and correction, which can be easily customized and deployed to solve application-specific data quality problems.

Emerging applications raise several challenges to build a generalized data cleaning platform including: **Heterogeneity:** Business and dependency based quality rules are expressed in a large variety of formats and languages from rigorous expressions (*e.g.,* functional dependencies), to plain natural language rules enforced by code embedded in the application logic itself (as in many practical scenarios). Such diversified semantics makes it difficult to create one uniform system to accept heterogeneous quality rules and enforce them on the data. **Interdependency:** Data cleaning algorithms are normally designed for one specific type of rules. [4] shows that interaction between two types of rules (CFDs and MDs) may produce higher quality repairs than treating them independently. However, the problem related to the interaction of more diversified types of rules is far from being solved. **Deployment and extensibility:** Although many algorithms and techniques have been proposed for data cleaning [1, 2, 4, 9], it is difficult to download one of them and run it on the data at hand without tedious customization. Adding to this difficulty is when users define new types of quality rules, or want to extend an existing system with their own cleaning solutions. **Metadata management and data custodians:** Several attempts have tackled the problem of including humans in the loop (*e.g.,* [5, 7, 9]). However, they only provide users with information in restrictive forms. In practice, the users need to understand much more meta-information *e.g.,* summarization or samples of data errors, lineage of data changes, and possible data repairs, before they can effectively guide any data cleaning process.

We present NADEEF [6], a data cleaning system that enables users to easily specify heterogeneous data quality rules specific to their applications, without worrying about how data errors are actually detected and repaired. In addition, NADEEF allows easy customization for domain experts. Moreover, NADEEF allows users to interactively manipulate data and metadata and includes them as first-class citizens in the data cleaning process. NADEEF leverages the separability of two main tasks: (1) isolating rule specification that uniformly defines *what* is wrong and (possibly) *how* to fix it; and (2) developing a core that holistically applies these routines to handle the detection and repairing of data errors. We will demonstrate the different features of NADEEF and how it addresses the challenges mentioned above. In particular, we present a data quality dashboard that exploits collected metadata and provides information such as error summarization and error distribution, for the users to easily interact with the system.

2. THE NADEEF ARCHITECTURE

Figure 1 illustrates the architecture of NADEEF. Overall, NADEEF consists of the following components: *data loader*, *rule collector*, *detection and cleaning core*, *metadata management*, and *data quality dashboard.*

[*] Work done while interning at QCRI.

Proceedings of the VLDB Endowment, Vol. 6, No. 12
Copyright 2013 VLDB Endowment 2150-8097/13/10... $ 10.00.

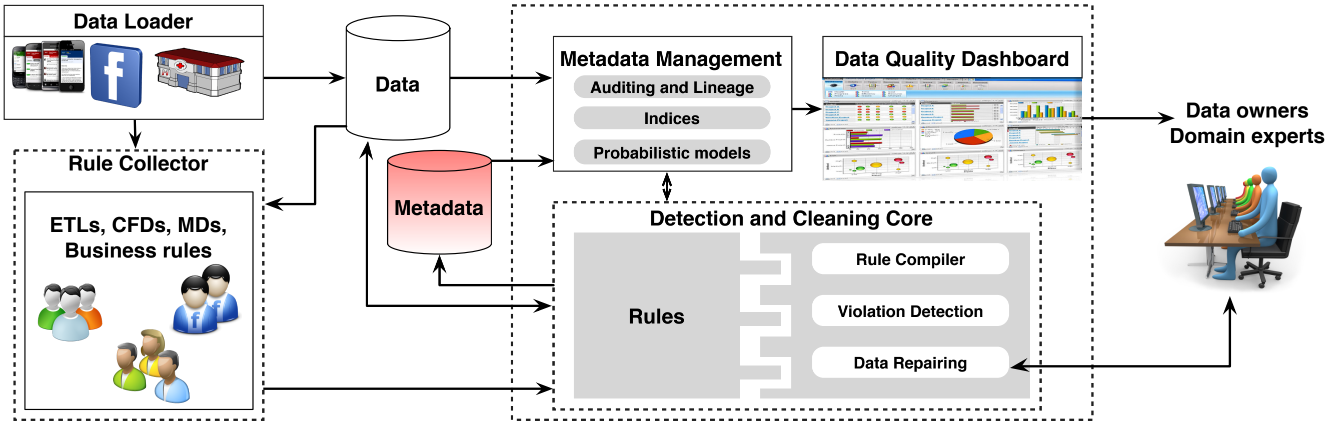
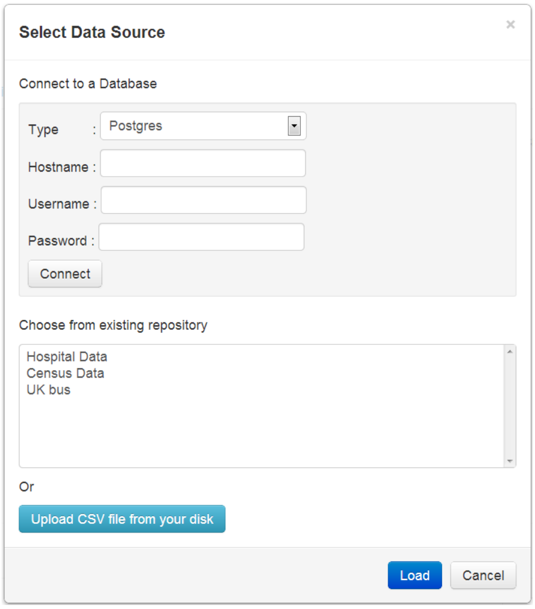


Figure 1: Architecture of NADEEF



Select Data Source

Connect to a Database

Type : Postgres

Hostname :

Username :

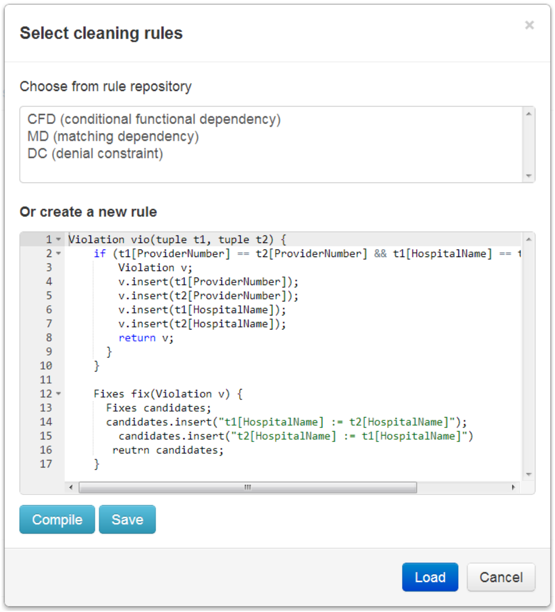
Password :

Choose from existing repository

Hospital Data
Census Data
UK bus

Or

Figure 2: Initialization of NADEEF



Select cleaning rules

Choose from rule repository

CFD (conditional functional dependency)
MD (matching dependency)
DC (denial constraint)

Or create a new rule

```
Violation vio(tuple t1, tuple t2) {
    if (t1[ProviderNumber] == t2[ProviderNumber] && t1[HospitalName] == t
        Violation v;
        v.insert(t1[ProviderNumber]);
        v.insert(t2[ProviderNumber]);
        v.insert(t1[HospitalName]);
        v.insert(t2[HospitalName]);
        return v;
      }
    }

    Fixes fix(Violation v) {
      Fixes candidates;
      candidates.insert("t1[HospitalName] := t2[HospitalName]");
        candidates.insert("t2[HospitalName] := t1[HospitalName]")
       reutrn candidates;
    }
```

Figure 3: Data quality rule management

NADEEF works as follows. It first collects data and heterogeneous rules from the users. The rule compiler then compiles these heterogeneous rules into homogeneous constructs. Next, the violation detection module identifies what data is erroneous and possible ways to repair them, based on user provided rules. The data repairing module fixes the detected errors by treating all the provided rules holistically. Note that data updates may trigger new errors to be detected. As the iterative detection and repairing process progresses, NADEEF collects and manages metadata related to its different modules. A live data quality dashboard exploits these metadata and visualizes information to users to interact with the system. In the following, we discuss these components in detail.

Rule collector. It collects user-specified rules such as ETL rules, CFDs (FDs), MDs, and other customized rules.

Detection and cleaning core. It contains three components: *rule compiler*, *violation detection* and *data repairing*.

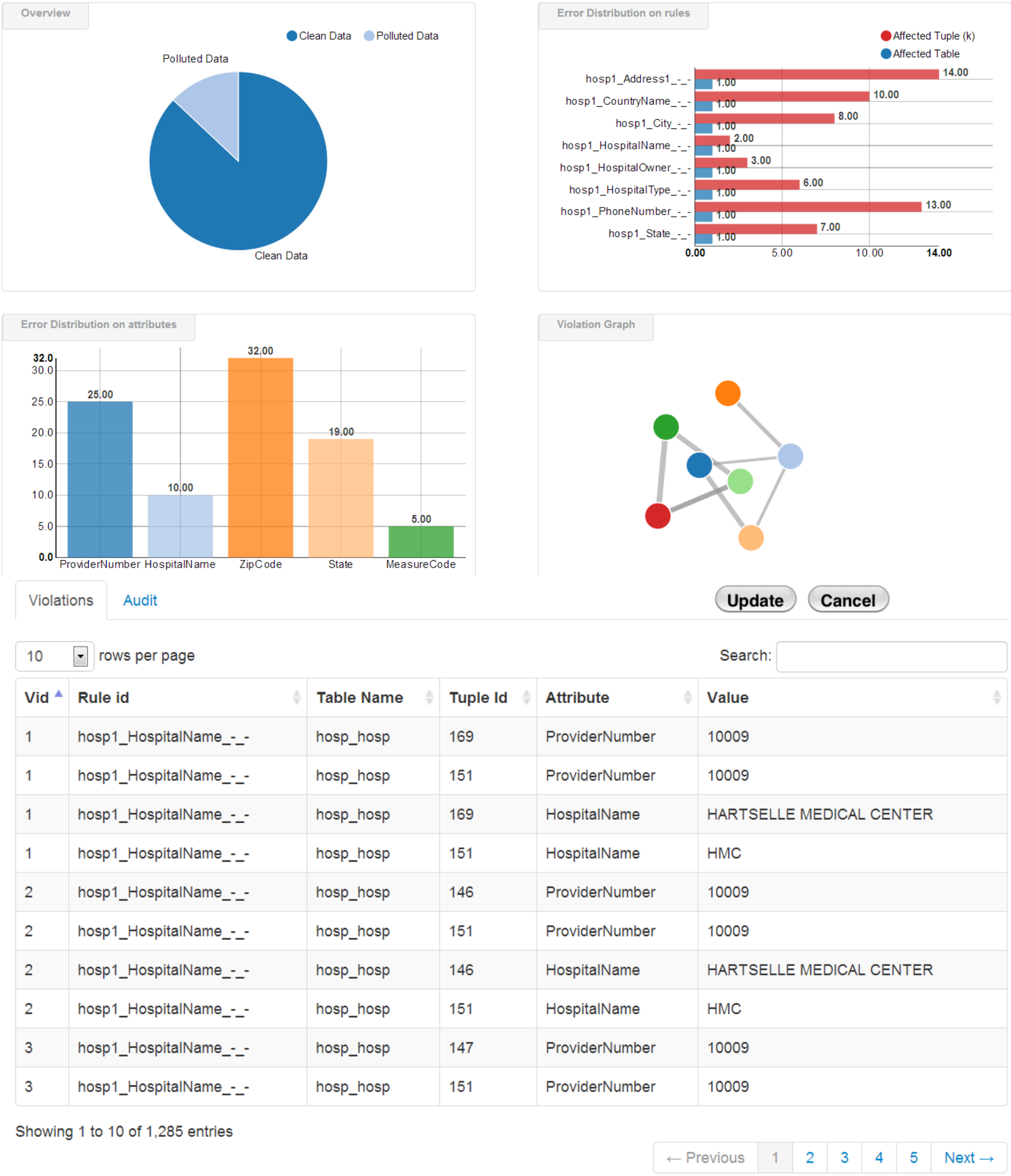
(i) Rule compiler. This module compiles all heterogeneous rules and manages them in a unified format.

(ii) Violation detection. This module takes the data and the compiled rules as input, and computes a set of data errors.

(iii) Data repairing. This module encapsulates holistic repairing algorithms that take violations as input, and compute a set of data repairs. By default, we use the algorithm [1], which can also be overwritten by *e.g.,* [2]. Moreover, this module may interact with domain experts through the *data quality dashboard* to achieve higher quality repairs.

Metadata management and data quality dashboard. Metadata management is to keep full lineage information about data changes, the order of changes, as well as maintaining indices to support efficient metadata operations. The data quality dashboard helps the users understand the health of the data through data quality indicators, as well as summarized, sampled or ranked data errors presented in live graphs. This facilitates the solicitation of users' feedback for data repairing.

Rule specification. We use the term *cell* to denote a combination of a tuple and an attribute of a table. The unified *programming interface* to define the semantics of data errors and possible ways to fix them is as follows.



Vid	Rule id	Table Name	Tuple Id	Attribute	Value
1	hosp1_HospitalName_-_-	hosp_hosp	169	ProviderNumber	10009
1	hosp1_HospitalName_-_-	hosp_hosp	151	ProviderNumber	10009
1	hosp1_HospitalName_-_-	hosp_hosp	169	HospitalName	HARTSELLE MEDICAL CENTER
1	hosp1_HospitalName_-_-	hosp_hosp	151	HospitalName	HMC
2	hosp1_HospitalName_-_-	hosp_hosp	146	ProviderNumber	10009
2	hosp1_HospitalName_-_-	hosp_hosp	151	ProviderNumber	10009
2	hosp1_HospitalName_-_-	hosp_hosp	146	HospitalName	HARTSELLE MEDICAL CENTER
2	hosp1_HospitalName_-_-	hosp_hosp	151	HospitalName	HMC
3	hosp1_HospitalName_-_-	hosp_hosp	147	ProviderNumber	10009
3	hosp1_HospitalName_-_-	hosp_hosp	151	ProviderNumber	10009

Showing 1 to 10 of 1,285 entries

Figure 4: Data quality dashboard

```
class Rule {
    set⟨cell⟩ vio (Tuple $s_1$) { return $\emptyset$ };
    set⟨cell⟩ vio (Tuple $s_1$, Tuple $s_2$) { return $\emptyset$ };
    set⟨Expression⟩ fix (set⟨cell⟩) { return $\emptyset$ };
}    /* end of class definition */
```

(1) vio(s) takes a single tuple s as input, and returns a set of problematic cells. By default, it returns an empty set.

(2) vio(s_1, s_2) takes two tuples s_1, s_2 as input, and returns a set of problematic cells. By default, it returns an empty set. Note that s_1, s_2 can come from either the same relation or two different relations.

(3) fix *(set⟨cell⟩)* takes a nonempty set of problematic cells as input, and returns a set of suggested expressions to repair these data errors.

The error detection function vio() expresses the rule's static semantics (*what* is wrong) while the function fix() reflects its dynamic semantics (*how* to repair errors). The presence of function fix() is optional, and its absence indicates that the users are not clear about the dynamic semantics of the rule. Here, a *violation* is a nonempty set V of *cells* that are returned by the function vio(s) or vio(s_1, s_2) of a data quality rule. Intuitively, in a violation, at least one of the cells is erroneous and should be modified.

A *candidate fix* F is a conjunction of expressions of the form "$c := x$", where c is a cell, and x is either a constant value or another cell. Intuitively, a candidate fix F is a set of expressions on a violation V, such that to resolve violation V, the modifications suggested by the expressions of F must be taken together. That is, to resolve a violation, more than one cell may have to be changed.

Similar to what most automatic data cleaning methods use to make their decision, we adopt minimality, *i.e.,* to compute an instance that repairs a database while incurring the least cost in terms of fixing operations. Since the users can plugin their own repairing algorithms, they can also use their own decision criteria for cleaning.

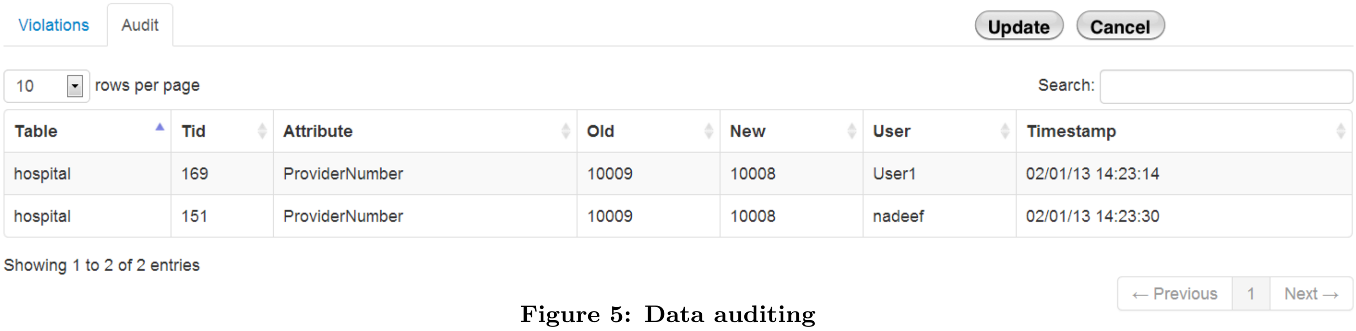


Table	Tid	Attribute	Old	New	User	Timestamp
hospital	169	ProviderNumber	10009	10008	User1	02/01/13 14:23:14
hospital	151	ProviderNumber	10009	10008	nadeef	02/01/13 14:23:30

Showing 1 to 2 of 2 entries

Figure 5: Data auditing

3. DEMONSTRATION OVERVIEW

In this demonstration, we will use real-life datasets to highlight several features of NADEEF and show the following: (1) how to easily prepare data; (2) how to specify multiple quality rules with the aid of a Web interface; (3) how the data quality dashboard can help users understand the health of data; (4) how the users can interact with NADEEF to easily correct errors; and (5) how to keep track of which attributes are corrected and where the correct values come from, after data has been updated.

(1) Data initialization. The users are required to configure a database instance (Fig. 2), by specifying a database connection with database type, hostname, username and password; and selecting the data sources, which can be either a loaded dataset, or new datasets to be uploaded.

(2) Data quality rule specification. Figure 3 displays the user interface for specifying and managing data quality rules. The users can either (a) load rules using rule classes *e.g.,* CFDs, MDs or DCs predefined using the programming interface; or (b) implement a customized rule by writing functions based on our programming interface in a few lines of code. The user interface will examine and report syntax errors of user implemented code. After both the data $\mathcal{D}$ and the data quality rules Σ are loaded, NADEEF first compiles all rules into the unified constructs conforming to our programming interface. It then detects all violations *w.r.t.* $\mathcal{D}$ and Σ, where each violation is a set of cells. These violations are stored in one violation table.

(3) Dashboard. We have implemented four graphs for easy interaction with the users, depicted in Fig. 4.

(a) *Overview*: This is to indicate the amount of data that is either involved in violations or is considered clean.

(b) *Error distribution on rules*: This shows (*i*) how many violations are detected for each rule, and (*ii*) how many data tables are involved.

(c) *Error distribution on attributes*: This shows the number of values that are involved in violations for each attribute. This is to reflect the *dirtiness* relative to various attributes.

(d) *Violation graph*: We build a violation graph to reflect the violations *w.r.t.* each data quality rule. In this graph, each node represents a rule. There is an edge between two nodes indicating that there are common cells that are involved in the violations for each rule (*i.e.,* node). The thickness of an edge indicates the number of common cells involved in the violation over both rules (*i.e.,* two ends of the edge).

For each of the above graphs (a-d), the users can drill down to see the corresponding part in the violation table, as shown in the bottom of Fig. 4. Moreover, users can specify predicates using "search"to select the data that they want to further explore.

(4) User interaction. By leveraging the data quality dashboard, NADEEF allows easy interaction with the users. As shown in Fig. 4, users can easily switch between different visualization modalities and identify errors based on their expertise and knowledge about the data. When such errors are found, the users can correct them directly on the data. They can also cancel their updates if needed. Moreover, the users can invoke default data repairing algorithms implemented in NADEEF to heuristically repair data.

(5) Data auditing. NADEEF provides a data auditing facility such that after data updates, the users may inspect the different changes made to the data. For example, Figure 5 shows two updates. Here, the first five attributes identify the updated cells, the attribute User identifies who changed the data, either a specific user or NADEEF, and Timestamp specifies when that update was committed to the database.

Summary. This demonstration aims at exhibiting the features of NADEEF. We focus on: (*i*) The isolation between a programming interface and core algorithms, such that users can easily define and manage heterogeneous data quality rules (See (2) above); (*ii*) The data quality dashboard that can help users to effectively interact with the system to inspect errors ((3) above), and to correct errors ((4) above); (*iii*) Data auditing such that the users can understand better what changes have been made to the data ((5) above).

4. REFERENCES

[1] 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.

[2] X. Chu, P. Papotti, and I. Ilyas. Holistic data cleaning: Put violations into context. In *ICDE*, 2013.

[3] Experian QAS Inc. `http://www.qas.com/`.

[4] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Interaction between record matching and data repairing. In *SIGMOD*, 2011.

[5] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu. Towards certain fixes with editing rules and master data. *VLDB J.*, 21(2), 2012.

[6] Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed Elmagarmid, Ihab F. Ilyas, Mourad Ouzzani, and Nan Tang. NADEEF: a commodity data cleaning system. In *SIGMOD*, 2013.

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

[8] N. Swartz. Gartner warns firms of 'dirty data'. *Information Management Journal*, 41(3), 2007.

[9] M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas. Guided data repair. *PVLDB*, 4(5), 2011.