Data Auditor: Exploring Data Quality and Semantics using Pattern Tableaux

Lukasz Golab, Howard Karloff, Flip Korn and Divesh Srivastava
AT&T Labs - Research
180 Park Avenue, Florham Park NJ, 07932, USA
{lgolab, howard, flip, divesh}@research.att.com

ABSTRACT

We present Data Auditor, a tool for exploring data quality and data semantics. Given a rule or an integrity constraint and a target relation, Data Auditor computes *pattern tableaux*, which concisely summarize subsets of the relation that (mostly) satisfy or (mostly) fail the constraint. This paper describes 1) the architecture and user interface of Data Auditor, 2) the supported constraints for testing data consistency and completeness, 3) the heuristics used by Data Auditor to "tune" a given constraint or its associated parameters for better fit with the data, and 4) several demonstration scenarios. using real data sets.

1. INTRODUCTION

Large-scale databases and data warehouses often appear to be haphazardly thrown together, owing to a variety of reasons from an inadequate understanding of data semantics (leading to poor schema design), to evolution of the database over time (due to integrating new sources) to error-prone data collection. Understanding the semantics of such data is useful for making sense of analysis results as well as detecting data quality problems. Recently, new integrity constraints have been proposed for capturing certain semantics of *data* (rather than schema). These include Conditional Functional Dependencies (CFDs) [4], Conditional Inclusion Dependencies (CIDs) [1], and Conditional Sequential Dependencies (CSDs) [6]. The key concept behind these constraints is the notion of *conditioning*: rather than requiring the constraint to hold over the entire relation, it need only be satisfied over conditioned subsets of the data. We view these constraints as rules which are capable of expressing consistency and completeness properties of the data.

This paper describes Data Auditor, a tool for exploring data semantics using conditional constraints. Data Auditor allows users to "try out" various types of constraints (including those listed above, and many more) to see if they hold or fail to a specified degree at least on some fraction of the data. Given a constraint, Data Auditor computes a *pattern tableau* that meaningfully summarizes the satisfying or failing subsets. Each pattern in a tableau may identify temporal intervals within the data or attribute values that most (but not necessarily all) satisfying or violating tuples have in common.

In addition to discovering a pattern tableau for a user-specified constraint and associated parameters, Data Auditor attempts to "tune" the constraint and its parameters for better fit with the data. One general tuning method decreases the specified degree to which the constraint needs to be satisfied, which may allow a larger fraction of the data and/or more meaningful subsets to be reported in the tableau. Data Auditor also supports many constraint-specific tuning strategies; for example, it may suggest to "relax" a functional dependency by removing an attribute from its right-hand-side if doing so gives a better representation of the underlying data semantics.

A great deal of work exists on data quality and data cleaning, and many systems have been proposed in this area, including Ajax [5], Bellman [2] and Potter's Wheel [9]. Data Auditor belongs to the class of constraint-driven systems, which also includes SEMANDAQ [3] and StreamClean [8]. SEMANDAQ employs CFDs and pattern tableaux to analyze data quality. It assumes that pattern tableaux are specified by the user, and focuses of on identifying violating tuples and suggesting ways to repair them. StreamClean focuses on constraint-based data cleaning and automatic error correction, and it requires each constraint to hold on the whole data set. Data Auditor complements tools like SEMANDAQ and StreamClean which assume that the data semantics are known apriori. By automatically discovering and tuning pattern tableaux for a variety of constraints, our system can provide a quick and user-friendly overview of the semantics and quality of the data.

2. OVERVIEW OF DATA AUDITOR

Figure 1 presents the architecture of Data Auditor. The Web-based user interface consists of a set of parameter screens, one for each supported constraint type (for example, one parameter may be the extent to which the subsets reported in the tableau must satisfy the constraint). Data Auditor then executes one of several algorithms (based on recent tableau discovery research [6, 7]) to efficiently generate concise tableaux for the selected constraint and associated parameters. The user receives a page containing the discovered tableau, as well as some *tuning* suggestions on how to "tweak" some of the parameters or the constraint itself in order to obtain a more concise tableau. Clicking on a particular pattern reveals more information about it, while clicking on a tuning suggestion displays the tableau for the tweaked constraint/parameters.

2.1 Supported Constraints

Data Auditor supports CIDs, CFDs, CSDs [6], as well as *Predicate Constraints* that specify conditions which should be satisfied by every tuple in a given relation. Functional and inclusion dependencies are well known; we describe the others below.

Our predicate constraints are of the form $\forall t \in R, c \rightarrow p$, which we write as:
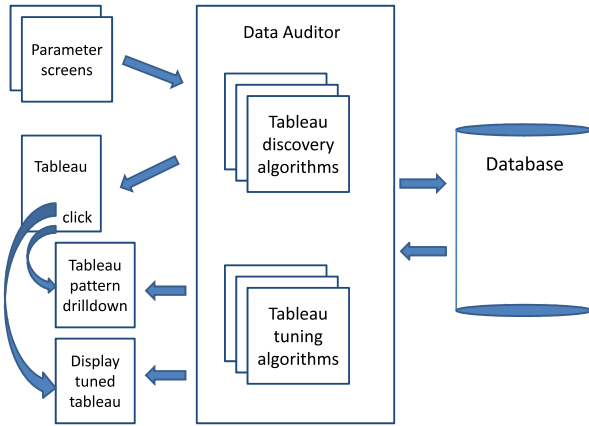
Figure 1: Overview of Data Auditor

Table 1: Example ROUTER_COUNTS table

name	location	time	num_responses
router1	New York	10:00	0
router2	New York	10:00	0
router3	Chicago	10:00	1
router4	Chicago	10:00	1
router1	New York	10:05	1
router2	New York	10:05	0
router3	Chicago	10:05	1
router4	Chicago	10:05	1
router1	New York	10:10	1
router2	New York	10:10	1
router3	Chicago	10:10	1
router4	Chicago	10:10	1
router1	New York	10:15	0
router2	New York	10:15	1
router3	Chicago	10:15	0
router4	Chicago	10:15	0

```
FOREACH t in R [WHERE c] ASSERT p
```

Here, t is a tuple, R is a relation, and c and p are predicates of any form allowed in the SQL `WHERE` clause. The meaning of the constraint is that if a tuple satisfies predicate c, it must also satisfy p; if c is not specified, then every tuple is expected to satisfy p. These types of constraints are useful for asserting predicates on the attribute values of every tuple (data consistency) and for asserting the existence of tuples (data completeness). For example, suppose that table `TICKETS` stores customer trouble tickets, with ticket id being the primary key and timestamp being the ticket submission time, and table `RESOLUTIONS` contains a row for every ticket id that has been resolved with its timestamp denoting the resolution time. The following constraint specifies that all trouble tickets must be resolved within 24 hours:

```
FOREACH t in TICKETS
ASSERT EXISTS (
  SELECT * FROM RESOLUTIONS u
  WHERE t.ticket_id = u.ticket_id
  AND u.timestamp >= t.timestamp
  AND u.timestamp - t.timestamp <= 24 )
```

Note that these predicate constraints can express functional and inclusion dependencies. For convenience, Data Auditor provides separate parameter screens for functional and inclusion dependencies so that users do not have to write them in predicate form.

Conditional Sequential dependencies (CSDs) have recently been proposed to express ordering properties in timestamped data such as the expected periodicity. A CSD takes two integer parameters, G_1 and G_2, and asserts that the "gaps" between the timestamps of consecutive tuples must be no less than G_1 and no more than G_2 (we allow G_1 to be as small as zero and G_2 to be as large as infinity). Here, an obvious tuning strategy that Data Auditor supports is to reduce G_1 and/or increase G_2.

In practice, constraints are rarely satisfied exactly. We employ the notion of *confidence* to measure the extent to which a relation or a data set satisfies a constraint. For predicate constraints, the confidence is defined simply as the fraction of rows of R satisfying c that additionally satisfy p. For CFDs and CSDs, we use the definitions from [7] and [6], respectively.

2.2 Pattern Tableaux

In addition to approximate constraint satisfaction, real data are often heterogeneous in the sense that certain subsets may satisfy a given constraint with very high (or very low) confidence. We employ pattern tableaux to summarize this valuable information. We discuss tableaux for predicate constraints and CFDs below (see [6] for details on tableaux for CSDs). Consider a set $A = a_1, a_2, \ldots, a_j$ of *conditioning attributes*, chosen from amongst the attributes in a relation. A pattern tableau consists of a set of patterns over A, each containing j symbols, one for each conditioning attribute. Each symbol is either a value in the corresponding attribute's domain or a special "wildcard" symbol '-'. Let $p_i[a_j]$ denote the symbol corresponding to the jth conditioning attribute of the ith pattern, and let $t[a_j]$ be the value of the jth conditioning attribute of a tuple t. A tuple t is said to *match* a pattern p_i if, for each a_j in A, either $p_i[a_j] = $ '-' or $t[a_j] = p_i[a_j]$. Note that the pattern consisting of only the '-' symbols matches the entire relation.

To compute a tableau, the user must specify the constraint itself, a set of conditioning attributes A, and two threshold parameters, $\hat{s}$ and $\hat{c}$ (for CFDs, we assume that A consists of its left-hand side attributes). The output is a tableau over A with (almost) as few patterns as possible whose union covers a fraction of $\hat{s}$ of the data, each of which has a confidence of at least $\hat{c}$. We refer to such a tableau as a *hold tableau* since it summarizes subsets on which the constraint holds. The user may also request subsets on which the constraint fails, in which case Data Auditor generates a *fail tableau*, where each pattern has confidence below $\hat{c}$.

We now give an example from the network monitoring domain. Suppose that each router within a network is expected to respond to a status query in every five-minute time bucket. Table 1 shows an example ROUTER_COUNTS relation that counts the number of responses per router in each five-minute period. One way to measure the completeness of these data is to try the following constraint.

```
FOREACH t in ROUTER_COUNTS
ASSERT num_responses > 0
```

The confidence of this constraint on Table 1 is $\frac{10}{16}$ since ten of 16 rows satisfy the given predicate.

Now suppose that we want to generate a *fail* tableau using the conditioning attribute set {`name`, `location`, `time`}, with a confidence upper bound $\hat{c} = 0.25$, and support (coverage) threshold $\hat{s} = 0.4$. This tableau is shown in Table 2, along with the confidence of each pattern and the number of tuples that match it. Note that this tableau summarizes subsets of ROUTER_COUNTS that are responding, on average, 25 percent of the time. For example, the first pattern has a confidence of 0.25

Table 2: Tableau for ROUTER_CPU_COUNTS

name	location	time	conf.	matches
-	-	10:15	0.25	4
-	New York	10:00	0	2
router2	-	10:05	0	1

since it matches the four tuples with `time=10:15`, of which only one has `num_responses > 0`. As a result, this fail tableau is a useful tool for summarizing the "worst offenders" of a constraint, and is easier to interpret than a (possibly very long) list of all violations. In fact, one can generate several fail tableaux with different $\hat{c}$ values to discover subsets with varying degrees of violations. Additionally, by changing $\hat{c}$ to be a lower bound, say, at least 0.9, the resulting *hold* tableau identifies "well behaved" subsets that satisfy the constraint with high confidence. For example, the pattern (- - 10:10) may be identified in such a tableau, which has a confidence of 1 since no responses were missing at time 10:10.

3. DEMONSTRATION SCENARIOS

So far, we have given examples of predicate constraints on network monitoring data. Many other constraints and data sets will be demonstrated, among them CFDs on sales data obtained from `amazon.com`. This data set contains 300,000 records, including fields such as `type`, `itemid`, `title`, `price`, `vat`, `quantity`, `userid`, `street`, `city`, `areacode`, `region`, `country`, `zip`. We hypothesize CFDs that hold in this data set. We start with the attribute set {`type`, `title`, `country`} and hypothesize that it strongly functionally determines {`price`, `vat`, `id`} on many subsets. (`type` and `title` together distinguish most items, and sales within the same country should have the same price and VAT charged.) On the full data, the FD $\{type, title, country\} \rightarrow \{price, vat, id\}$ has confidence 0.72.

Figure 2 displays the parameter screen for CFDs. We specify the target table, the antecedent (left-hand-side) and consequent (right-hand-side) of the FD, the confidence threshold of (to generate a fail tableau, we write a minus in front of the confidence threshold), and the coverage (support) threshold. We also need to choose a tuning option: "none", "confidence" (i.e., generate a second tableau using a slightly lower confidence threshold), or "confidence and attributes" (i.e., in addition to tweaking the confidence, generate tableaux for similar FDs by removing an attribute from the consequent or adding an attribute to the antecedent). There is also an optional filter parameter, which serves a similar purpose to the c predicate in predicate constraints. That is, Data Auditor will compute a tableau over only those tuples from the specified table which satisfy c. Other types of constraints have slightly different parameter screens, depending on the necessary parameters.

Given the parameters in Figure 2, Data Auditor computes a tableau shown in Figure 3 (in the background), which contains 40 patterns. We can click on an individual pattern and drill down into more specific patterns by expanding any wildcards into all possible constants. In the foreground of Figure 3, we show the result of clicking on the tableau pattern "-,-,GER", which lists all the possible item types and item titles purchased in Germany (GER). Another way of interactively exploring the data is to choose a pattern, say "-,-,GER", and generate another tableau using this pattern as a filter parameter (recall Figure 2). This new tableau summarizes subsets of items purchased in Germany that satisfy the related FD $\{type, title\} \rightarrow \{price, vat, id\}$.
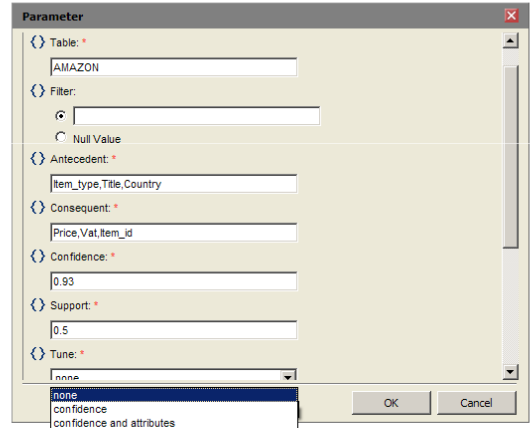
Figure 2: Specifying a Functional Dependency in Data Auditor

The user can also view *tuned* tableaux that were generated by Data Auditor in parallel with the original tableau. Links to tuned tableaux appear at the bottom of Figure 3, along with their sizes. The first link leads to a tuned tableau that was created by lowering the confidence threshold to 0.88. This tuned tableau only contains two patterns, meaning that we can summarize this data set much better by decreasing the extent to which the specified FD must hold. On the other hand, the second link indicates that using a consequent of Vat and Item id (but not Price), is not helpful since the resulting tableau has the same size (40 patterns) as the original tableau. However, the third link shows that removing Vat from the consequent, which changes the original FD to $\{type, title\} \rightarrow \{price, id\}$, allows the new FD to fit much better since the resulting tableau, illustrated in Figure 4, only contains two patterns.

We can test other interesting rules on this data set and obtain tuning suggestions, e.g., that zip codes functionally determine geographic attributes. For example, starting with the FD $\{country, zip\} \rightarrow \{areacode, city, region, street\}$, Data Auditor suggests that we remove both `city` and `areacode` from the consequent to improve the fit.

4. REFERENCES

[1] L. Bravo, W. Fan, and S. Ma. Extending dependencies with conditions. *VLDB 2007*, 243-254.

[2] T. Dasu, T. Johnson, S. Muthukrishnan and V. Shkapenyuk. Mining database structure; or, how to build a data quality browser. *SIGMOD 2002*, 240-251.

[3] W. Fan, F. Geerts and X. Jia. Semandaq: A Data Quality System Based on Conditional Functional Dependencies. *PVLDB*, 1(2): 1460-1463, 2008.

[4] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *TODS*, 33(2):1-48, 2008.

[5] H. Galhardas, D. Florescu, D. Shasa, E. Simon and C.-A. Saita. Declarative data cleaning: language, model, and algorithms. *VLDB 2001*, 371-380.

[6] L. Golab, H. Karloff, F. Korn, A. Saha, and D. Srivastava. Sequential dependencies. *VLDB 2009*.

[7] L. Golab, H. Karloff, F. Korn, D. Srivastava, and B. Yu. On generating near-optimal tableaux for conditional functional dependencies. *PVLDB*, 1(1):376-390, 2008.

[8] N. Khoussainova, M. Balazinska, and D. Suciu. Towards correcting input data errors probabilistically using integrity constraints. *MobiDE 2006*, 43-50.

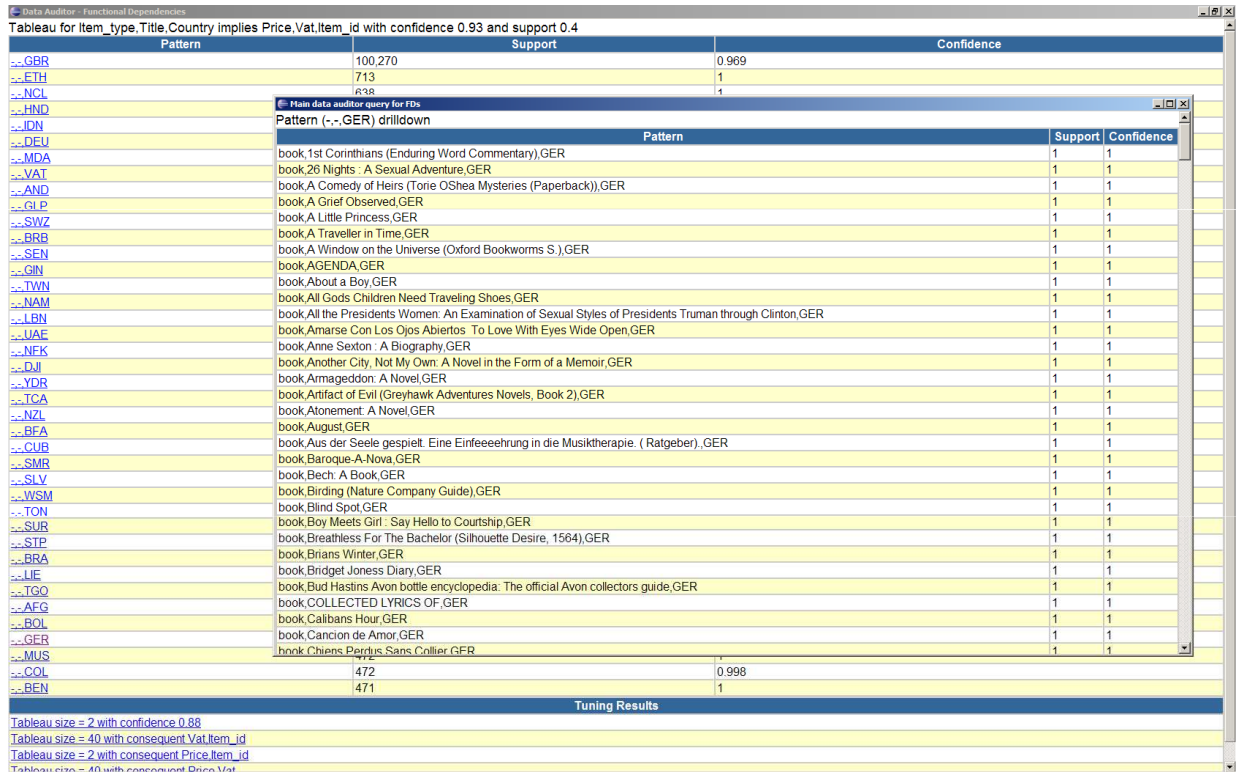[9] V. Raman and J. Hellerstein. Potter's wheel: An interactive data cleaning system. *VLDB 2001*, 381-390.
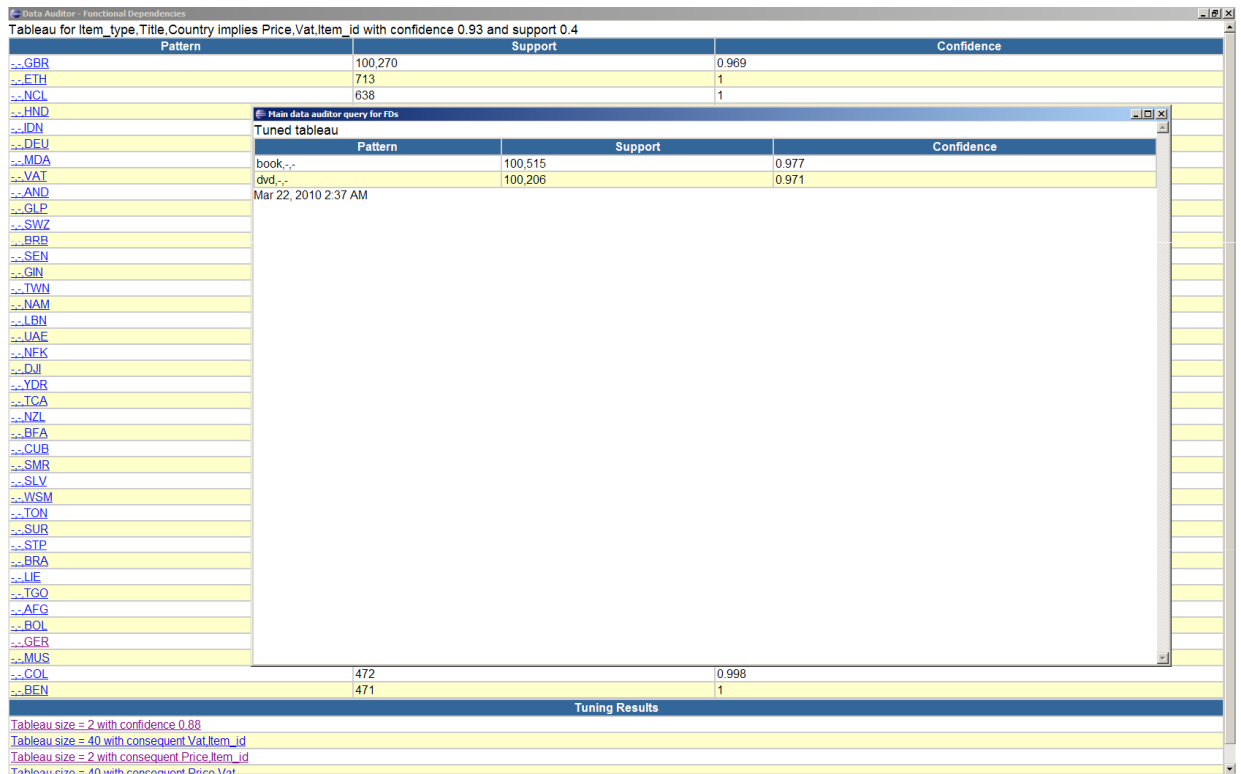
Figure 3: Generating a tableau in Data Auditor

Figure 4: Tuning a tableau in Data Auditor