Discovering Pattern Tableaux for Data Quality Analysis: a Case Study

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

ABSTRACT

In this paper, we present a case study that illustrates the utility of *pattern tableau* discovery for data quality analysis. Given a user-supplied integrity constraint, such as a boolean predicate expected to be satisfied by every tuple, a functional dependency, or an inclusion dependency, a pattern tableau is a concise summary of subsets of the data that satisfy or fail the constraint. We describe Data Auditor—our system for automatic tableau discovery from data—and we give real-life examples of characterizing data quality in a network monitoring database used by a large Internet Service Provider.

1. INTRODUCTION

This paper presents a case study of pattern-tableau-driven data quality analysis. Given a user-supplied constraint over one or more relations, the goal of this type of analysis is to automatically generate a concise and easy-to-understand summary of tuples that satisfy or fail the constraint (rather than simply listing all the satisfying and violating tuples, which may not be easy to interpret by the user and may not reveal any interesting patterns in the data). The produced summaries, referred to as *pattern tableaux*, specify attribute values that most (but not necessarily all) satisfying or violating tuples have in common. We will motivate and describe various types of useful constraints, give a brief overview of the Data Auditor tool for automatic tableau generation from data [14], and show the utility of our approach on a network monitoring database used by a large Internet Service Provider (ISP).

While our approach is applicable to any type of data, this case study focuses on monitoring databases, which maintain information about computer systems such as IP networks, Web servers, sensor networks, or cloud computing clusters. We illustrate a network monitoring example in Figure 1. For the purposes of this paper, a network consists of routers, each having at least one interface, and links that connect pairs of interfaces. A network monitoring database contains configuration tables that describe the devices in the network and their properties (IP address, location, date of deployment in the network, router type, model and operating system version, link capacity, etc.). Configuration tables may change every day, as new equipment is added, existing equipment moved or upgraded, and obsolete equipment turned off. Additionally, measurement tables are associated with the equipment being monitored, among them system logs, user trouble tickets, router CPU and memory usage statistics, and bandwidth usage reports. Measurement tables are updated (appended to) by a polling mechanism that collects data from the network. In a large network, the polling may be done by a set of distributed pollers.

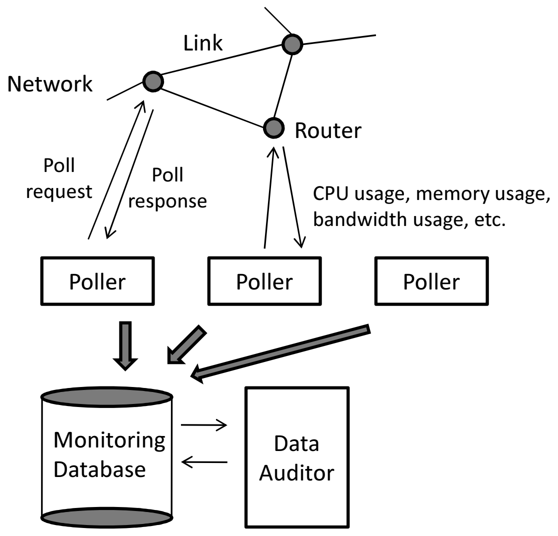

Figure 1: A network monitoring database.

Monitoring databases are prone to traditional data quality issues such as data entry errors (e.g., incorrect specifications entered for a new piece of equipment) and duplicates (e.g., different configuration entries for the same router). Additionally, malfunctioning devices may generate incorrect measurements or fail to respond to polls. In fact, even healthy devices may have incorrectly configured firewalls that block incoming poll requests. The polling mechanism and the network itself may also cause data quality problems: some devices may not be polled or may be polled too often by multiple pollers, poll requests and/or responses may be lost, etc. As a result, measurement tables may suffer from *incorrect*, *missing*, *duplicate* and *delayed* data.

Monitoring databases serve important business functions: troubleshooting customer problems, analyzing equipment failures, detecting malicious users, planning system upgrades, etc. In this paper, we show that generating pattern tableaux for user-specified integrity constraints is a useful technique for understanding monitoring data and characterizing its quality. We also show that tableau-driven data quality analysis can help diagnose "process quality" problems with the polling mechanism and problems with the network itself.

The remainder of this paper is organized as follows. Section 2 describes the types of constraints supported by our Data Auditor tool and the tableau generation algorithm. In Section 3, we demonstrate the utility of tableau-driven data quality analysis on a network monitoring database used by a large ISP. Section 4 discusses related work and Section 5 concludes the paper.

2. OVERVIEW OF DATA AUDITOR

2.1 Supported Constraints

Data Auditor supports constraints of the form $\forall t \in R, c \rightarrow p$. In this paper, we will use the following SQL-like syntax to express such constraints:

```
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. In a data warehouse setting, `R` may be a "base" table or a complex materialized view defined over one more more tables (or other views). The "`WHERE c`" part is optional.

For example, it is reasonable to assume that every CPU utilization poll in the `ROUTER_CPU` measurement table must have a value between zero and 100; violations may be caused by bugs in the router operating system function that records CPU usage:

```
FOREACH t IN ROUTER_CPU
ASSERT t.cpu_util >= 0 AND t.cpu_util <= 100
```

Note that there is no `WHERE` clause in this constraint (and in other examples in this section) since we want the `ASSERT` predicate to hold on the entire `ROUTER_CPU` table.

We can also express *Inclusion Dependencies*. If the `router_id` associated with every CPU poll in the `ROUTER_CPU` table must exist in the `ROUTERS` configuration table, we write:

```
FOREACH t in ROUTER_CPU
ASSERT EXISTS (
  SELECT *
  FROM ROUTERS u
  WHERE t.router_id = u.router_id )
```

Another useful constraint, particularly in monitoring databases, defines temporal correlations across related events. 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 )
```

Functional Dependencies (FDs) may also be expressed in our framework. To assert that router model functionally determines router manufacturer (`model` $\rightarrow$ `manufacturer`) in the configuration table `ROUTERS`, we write:

```
FOREACH t in ROUTERS
ASSERT NOT EXISTS (
  SELECT *
  FROM ROUTERS u
  WHERE t.model = u.model
  AND NOT (t.manufacturer = u.manufacturer))
```

2.2 Measuring the Confidence of a Constraint

We employ the notion of *confidence* to measure the extent to which a given relation (or a subset of it) satisfies a constraint. We define confidence as the fraction of rows of `R` (for which `c` is true) that satisfy the assertion `p`. We can compute the numerator of this quantity as

```
SELECT count(*) FROM R t WHERE c AND p
```

and the denominator as

```
SELECT count(*) FROM R t WHERE c
```

Other definitions of confidence, such as that used in [15] for FDs, may also be expressed in Data Auditor, so long as the corresponding SQL queries are provided.

The confidence of an integrity constraint may be thought of as a metric to characterize data quality. For example, consider the materialized view `ROUTER_CPU_COUNTS` illustrated in Table 1, which maintains the number of CPU polls returned by each router, uniquely identified by router name, in each five-minute period. This view is the result of a complex aggregation query (not shown) over `ROUTER_CPU`, which adds up the number of records in `ROUTER_CPU` for each router in each possible five-minute time bin, with a default aggregate value of zero for time bins with no polls received. This ensures that there are records in `ROUTER_CPU_COUNTS` for every 5-minute period throughout the day. Suppose that each row also contains the location of the given router. Suppose also that we require at least one poll from every router every five minutes (even if multiple polls arrive in a single 5-minute interval, we still require a poll in the next 5-minute interval). One way to measure the completeness of the CPU poll data is via the confidence of the following constraint:

```
FOREACH t in ROUTER_CPU_COUNTS
ASSERT t.num_polls > 0
```

In our example from Table 1, the confidence of this constraint is $\frac{10}{16}$, which indicates that, on average, 62 percent of the expected CPU polls (in terms of the five-minute time intervals with at least one poll) are there and 38 percent are missing.

2.3 Pattern Tableaux

Real data, especially monitoring data collected from a diverse system, are heterogeneous. In our network monitoring example, routers from different locations may be missing different fractions of polls, and any given router may behave differently over time. Therefore, in addition to reporting the confidence on the entire relation, it is useful to identify parts of the relation that satisfy the constraint (i.e., have high confidence) and parts that violate it (i.e., have low confidence). The idea behind tableau-driven data quality analysis (and Data Auditor) is to compactly summarize this valuable information.

Table 1: Example ROUTER_CPU_COUNTS table

name	location	time	num_polls
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

Following [9], we use *pattern tableaux* to encode subsets of a relation. Consider a set $A = a_1, a_2, \ldots, a_\ell$ of *conditioning attributes*, chosen from amongst the attributes in the relation. A pattern tableau consists of a set of patterns over A, each containing ℓ 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 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 $t[a_j] = p_i[a_j]$. The confidence of a pattern p_i is defined as the confidence of a sub-relation containing only those tuples that match p_i. Note that the pattern consisting of only the '-' symbols matches the entire relation.

The input to Data Auditor is a relation R (again, R could be a complex materialized view), a constraint (of the supported type), a set of conditioning attributes A (where $A \subseteq R$), a positive integer k, and a fraction $\hat{c}$, which is either a lower or an upper bound on the confidence of each tableau pattern on R. The output is a tableau over A with at most k patterns, each of which has a confidence of at least or at most $\hat{c}$, depending on the intended meaning of $\hat{c}$. As in [15], we refer to the former as a *hold tableau* since it summarizes subsets on which the constraint holds, and the latter as a *fail tableau*, which detects subsets on which the constraint fails.

The tableau construction step uses a generalized version of the *on-demand* algorithm from [15], which was originally proposed to generate tableaux only for FDs. Data Auditor first pre-computes the confidence and the number of matching tuples for each pattern that contains no wildcards (which can then be used to compute the confidence of patterns with wildcards). Each iteration of the algorithm then inserts into the tableau a pattern that (meets the required confidence threshold and) matches the most tuples that have not already been matched. This greedy heuristic attempts to produce the smallest possible tableau (having the fewest patterns) that matches the largest possible fraction of the relation. Thus, general patterns with wildcards are more likely to be included (provided that they have the appropriate confidence) than specific patterns that match a small fraction of the data.

Recall the `ROUTER_CPU_COUNTS` relation from Table 1. Suppose that `A = {name, location, time}`, $\hat{c}$ is an upper bound of 0.25, and $k = 5$. Table 2 shows a *fail* tableau for the following constraint, along with the confidence of each pattern and the number of tuples that match it (note the syntax for specifying the conditioning attributes, confidence, and maximum tableau size):

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

```
FOREACH t IN ROUTER_CPU_COUNTS
ASSERT t.num_polls > 0
TABLEAU t.name, t.location, t.time
CONF <= 0.25
MAX SIZE 5
```

This tableau only has three rows, two shy of `MAX SIZE`. However, there are no more patterns with confidence under 0.25 that match any tuples that have not already been matched, so the tableau generation algorithm terminates early[1]. Note that this tableau summarizes subsets of `ROUTER_CPU_COUNTS` that, on average, are returning at most 25 percent of the polls (i.e., at least 75 percent of the polls are missing). For example, the first pattern has a confidence of 0.25 since it matches the four tuples with `time=10:15`, of which only one has `num_polls > 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 `CONF` parameters to discover subsets with varying degrees of violations. For instance, the last two patterns in Table 2 represent subsets that are missing every poll (the confidence is zero). Additionally, by changing the `CONF` specification of the above constraint to, say, `>= 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 one since no polls were missing at time `10:10`. Also note that tableau patterns may overlap.

The choice of conditioning attributes is crucial to obtaining concise and informative tableaux—some attributes may be irrelevant to the constraint at hand. Fortunately, monitoring databases typically contain timestamp fields, which often produce interesting patterns, as well as attributes that naturally partition the devices being monitored into groups, such as router type or location. In future work, we plan to study automatic selection of conditioning attributes in more detail, e.g., by exploiting attribute correlations.

So far, we have discussed tableaux containing patterns with constants drawn from the attribute domains or the wildcard symbol '-'. Data Auditor also supports *range tableaux* [15], which, in addition to constants and wildcards, contain patterns with ranges of values of ordered attributes. Here, the definition of matching extends in the obvious way—a tuple matches the pattern if the value of the ordered attribute of the tuple falls within the range. Patterns with ranges often lead to smaller tableaux since a single range pattern may be used instead of separate patterns for different values in the range. In Table 3, we show a tableau for the same constraint as that in Table 2, whose second pattern includes a range on the `time` attribute. Note that this tableau is smaller than Table 2; the last two

[1]Note that the tableau generation algorithm always terminates, either after reaching the maximum tableau size bound, or when it runs out of patterns that meet the confidence threshold.

Table 3: Tableau for ROUTER_CPU_COUNTS with a range pattern

name	location	time	conf.	matches
-	-	10:15	0.25	4
-	New York	10:00 till 10:05	0.25	4

patterns from Table 2 are now captured by a single pattern in Table 3. By default, Data Auditor considers patterns with ranges for each ordered attribute in the conditioning attribute set.

3. CASE STUDIES

We now present examples of pattern-tableau-driven data quality analysis of a network monitoring database used by a large ISP. This database is maintained using the DataDepot warehouse system [12] and consists of a number of configuration tables, including `ROUTERS` and `INTERFACES`. Each router has a name, IP address, class (backbone router, customer router, etc.), location, etc. Each interface resides on a particular router, and has a name, IP address, function, capacity, etc. Measurement tables include `ROUTER_CPU` (poll responses with average CPU utilization per router per five-minute time period), `ROUTER_MEMORY` (likewise for average memory utilization), and many more. We also maintain a number of materialized views, among them `ROUTER_CPU_COUNTS` and `ROUTER_MEMORY_COUNTS`, which, similar to Table 1, count the number of polls returned by each router in each possible 5-minute time interval. Again, these views are computed via complex aggregation over the corresponding CPU and MEMORY base tables (in DataDepot, we expressed these views using EMF-SQL [17], which is an extension of SQL that supports complex grouping conditions). In total, this network monitoring database contains over 300 tables and materialized views, and ingests over 300 million new records per day.

Due to the proprietary nature of the data, we have anonymized the attribute values shown in the pattern tableaux in this section, and we do not display the number of matching tuples per pattern. Also, for brevity, we do not report any detailed performance results. The running time of the first step (pre-computing the confidence of patterns without wildcards) involves executing group-by aggregation queries on the tables involved in the constraint, and depends on the sizes of these tables. The second step (execution of the tableau generation algorithm) took on the order of one or two minutes at most for each of the examples in this section, when executed on our database application server.

3.1 Missing Polls

We begin by analyzing missing CPU data over a period of one week using the following constraint (from now on, we omit the `MAX SIZE` clause and display the first few rows of each tableau):

```
FOREACH t IN ROUTER_CPU_COUNTS
ASSERT t.num_polls > 0
TABLEAU t.name, t.class, t.hour_of_day, t.time
CONF <= 0.5
```

This constraint references the view `ROUTER_CPU_COUNTS`. That is, we want to verify that we have received at least one poll per router during each five-minute time interval of every day (rather than, say, examining the `ROUTER_CPU` base table to see if it has received at least 288 polls per router per day without investigating how the polls are spread out throughout the day). This is an important property to monitor because network analysts often need to look up router CPU usage (or other measurements) during a particular point in time and correlate with other network events that happened during this time; thus, we want the CPU polls to "cover" every possible 5-minute time interval.

Table 4: Tableau for missing CPU polls

name	class	hour of day	time	conf.
-	vpn	-	-	0.33
-	-	-	Sat 18:00 till Sat 20:00	0.05
-	backbone	17 till 20	-	0.3
-	backbone	-	Sat 21:00 till Sun 04:00	0

The conditioning attributes for this constraint are router `name`, router `class`, `hour_of_day` (an integer between 0 and 23 corresponding to the hour of the time attribute), and `time`. Adding hour of day to the conditioning attributes allows us to find daily recurring patterns in the missing data, if any. The confidence of this constraint on the entire view `ROUTER_CPU_COUNTS` is 87 percent (i.e., 13 percent of the polls are missing). The generated fail tableau, shown in Table 4, identifies subsets with high data loss (at least 50 percent). Note that both `hour_of_day` and `time` are ordered and give rise to patterns with ranges.

This tableau identifies several interesting patterns that may be presented to network technicians for further analysis. First, `vpn` routers appear to be missing two thirds of the CPU polls. Second, there appears to have been a problem on Saturday night that affected all routers (perhaps there was a network outage or a problem with the polling mechanism). Third, `backbone` routers seem to be affected by missing polls every day between 5pm and 8pm, and have not been polled at all (as indicated by the 0 percent confidence of the last pattern) between 9pm on Saturday and 4am on Sunday. Note that a concise pattern tableau is simpler to interpret than a list of all violations of the given constraints.

Note that tableau-driven data quality analysis requires the user to supply a constraint and is orthogonal to discovering all the constraints of a certain type that hold on a given table. In a large database, it may not be feasible to search for all the possible constraints. Instead, our approach is to use human input and domain knowledge in a constructive way. In the above example, with the help of domain knowledge, we posed a constraint that describes the completeness of the CPU measurement data. Given a particular constraint, we then focused on searching for subsets of the given view(s) in which this constraint is violated, which helped guide further analysis of these data by domain experts.

3.2 Correlated Missing Polls

Having learned that CPU polls are missing from certain routers at certain times of the day, we now hypothesize that missing CPU measurements may be related to missing memory usage measurements. Testing such a constraint (along with domain knowledge) may help us understand the root cause of data loss. For instance, if both CPU and memory polls were missing at a particular time, then the router and/or the entire polling mechanism may have been down. If only CPU or memory polls are missing, then the router may be selectively ignoring some types of poll requests or the polling mechanism may not be sending certain types of polls to the database.

Table 5: Tableau for correlated missing polls

name	class	time	conf.
-	-	Sat 18:00 till Sat 20:00	0.95
-	edge1	-	0.81
-	edge2	-	0.91
-	cust1	-	0.99

First, we test the constraint that if a CPU poll is missing in the view `ROUTER_CPU_COUNTS` (i.e., `num_polls = 0`) for some router during some 5-minute interval, then the corresponding memory poll in the view `ROUTER_MEMORY_COUNTS` should also be missing. We use the same week's worth of data as in the previous experiment. Note the use of the `WHERE` clause to restrict the scope of this constraint to missed CPU polls:

```
FOREACH t IN ROUTER_CPU_COUNTS
WHERE t.num_polls = 0
ASSERT EXISTS (
  SELECT * FROM ROUTER_MEMORY_COUNTS u
  WHERE t.name = u.name AND t.time = u.time
  AND u.num_polls = 0 )
TABLEAU t.name, t.class, t.time
CONF >= 0.8
```

Since the confidence threshold is at least 0.8, the resulting *hold* tableau denotes patterns where, on average, at least 80 percent of the missing CPU polls are associated with a missing memory poll. In this example, we no longer use `hour_of_day` as a conditioning attribute as it did not generate any interesting patterns.

The tableau is shown in Table 5. It includes the pattern

```
(- - Sat 18:00 till Sat 20:00)
```

as does the tableau in Table 4, suggesting that almost no polls of any kind were collected in this time period. The other patterns reveal that for several router classes, a dropped CPU poll usually co-occurs with a dropped memory poll.

Note that the constraint used in this example is not a typical data integrity constraint whose violations can be thought of as "dirty" data. Instead, this is a data exploration constraint that specifies some property of interest (i.e., correlated loss of CPU and memory polls), whose tableau indicates which parts of the relation satisfy the claimed property. Both data integrity and data exploration are important components of data quality analysis.

Next, we test the "opposite" constraint, on the same portion of data as before, which asserts that a missed CPU poll is *not* correlated with a missed memory poll in the same 5-minute time window, i.e.,

```
FOREACH t IN ROUTER_CPU_COUNTS
WHERE t.num_polls = 0
ASSERT EXISTS (
  SELECT * FROM ROUTER_MEMORY_COUNTS u
  WHERE t.name = u.name AND t.time = u.time
  AND u.num_polls > 0 )
TABLEAU t.name, t.class, t.time
CONF >= 0.8
```

The hold tableau is illustrated in Table 6. The first pattern suggests that backbone routers do not respond to CPU polls, e.g., due to a firmware bug that blocks CPU poll requests (but they do respond to memory polls). Also, it appears that the polling mechanism lost CPU polls, but not memory polls, early Sunday morning.

Table 6: tableau for uncorrelated missing polls

name	class	time	conf.
-	backbone	-	0.84
-	-	Sun 00:00 till Sun 03:00	0.82

Table 7: Tableau for duplicate memory polls

name	class	time	conf.
newyork-r25	-	Fri 17:00 till Fri 23:00	0.96
newyork-r31	-	Fri 17:00 till Fri 23:00	0.96
chicago-r07	-	Fri 17:00 till Fri 23:00	0.98

For brevity, we omit the results of a similar set of experiments that tests correlations between missing CPU and memory polls in "the other direction", i.e., checking the existence or absence of a CPU poll given that the corresponding memory poll is missing. Finally, note that Tables 4, 5 and 6 suggest possible causes of missed polls, which can then be verified and corrected by network engineers.

3.3 Duplicate Polls

Missing measurement data is clearly an important data quality problem. Duplicate or extraneous data are also of interest since reporting too many polls puts unnecessary overhead on the devices being monitored and on the monitoring infrastructure. We now investigate extraneous memory polls, defined as occurrences of multiple polls from the same router in the same 5-minute time interval, using the following constraint:

```
FOREACH t IN ROUTER_MEMORY_COUNTS
ASSERT t.num_polls <= 1
TABLEAU t.name, t.class, t.time
CONF <= 0.5
```

The *fail* tableau is shown in Table 7, using the same week's worth of data as in the previous experiments. It suggests that a group of routers were continuously double-polled during a six-hour window on Friday evening, perhaps due to a misconfiguration of multiple distributed pollers that led to overlapping poller coverage. However, not all routers were double-polled at that time, only those listed in the tableau (otherwise the entire pattern `(- - Fri 17:00 till Fri 23:00)` would appear in the tableau).

We then hypothesized that perhaps there exist instances of duplicate polls from other routers due to irregularly spaced polls. Such instances would satisfy a constraint which asserts that time intervals with double-polls are preceded or followed by time intervals with zero polls from a given router (we omit the exact specification of this constraint for brevity). However, we obtained empty hold tableaux when we tested this constraint on several different weeks of data from the `ROUTER_MEMORY_COUNTS` view. Clearly, data quality exploration can involve testing hypotheses that may not be true in the data.

3.4 Problems with Configuration Data

So far, defining missing and extraneous data was relatively straightforward because we knew (or were able to find out from

Table 8: Tableau for missing interfaces

date	class	network	conf.
01-26	-	-	0
03-15	-	-	0.08
04-01	-	-	0.09
-	bgp	-	0
02-28	-	Europe	0.25

domain experts) the expected polling frequency. We now analyze a problem in which it is not possible to identify missing data in one table without checking for the existence of a record in another table. In particular, we want to verify the completeness of the `INTERFACES` configuration table with the help of the `ROUTERS` configuration table, using a six-month excerpt of these two tables.

We know that when a new router is deployed, it is inserted into the `ROUTERS` table on the same day. Additionally, we hypothesize that at least one interface on this router must be added to the `INTERFACES` table on the same day—clearly, a router must have at least one active interface in order to route traffic. The corresponding Data Auditor constraint is:

```
FOREACH t IN ROUTERS
ASSERT EXISTS (
  SELECT * FROM INTERFACES u
  WHERE t.router = u.router
  AND t.date = u.date )
TABLEAU t.date, t.class, t.network
CONF <= 0.25
```

That is, if a router is added on a certain date, at least one of its interfaces must be added to the `INTERFACES` table with the same date. The conditioning attributes are date, router class, and the network to which the router belongs. Again, note that it is not possible to detect missing interfaces using the `INTERFACES` table alone; we must first check for new routers in the `ROUTERS` table.

The fail tableau is illustrated in Table 8. Note that a violation here means that either the interfaces for the new router were never added to the database, or they appear in the `INTERFACES` table, but their dates are not the same as the date on which the router was added (we can define a separate constraint to distinguish between these two cases).

The first three patterns in Table 8 reveal specific dates where nearly all the new routers that were added on those dates did not have any interfaces added to the `INTERFACES` table (or these interfaces have the wrong dates). According to the fourth pattern, all `bgp` routers have this problem, and, according to the last pattern, routers added to the `Europe` network on February 28 violate the given constraint (but routers added to other networks on that day do not since the pattern `(02-28 - -)` is not included).

In our final example, exploring the `ROUTERS` table using a functional dependency plus a corresponding tableau reveals data quality issues and interesting facts about the data. The FD we test asserts that router class functionally determines router model:

```
FOREACH t in ROUTERS
ASSERT NOT EXISTS (
  SELECT * FROM ROUTERS u
  WHERE t.class = u.class
  AND NOT (t.model = u.model) )
TABLEAU t.class
CONF >= 0.9
```

Table 9: Tableau for the FD class $\rightarrow$ model

class	conf.
class-5	1
Cisco super 5000	1
class-10	1
UNKNOWN	1

That is, we want to determine if there exist router classes for which one particular router model is used exclusively (with high confidence). We expect this constraint to be satisfied by only a few classes, if any, since it would be unusual for a large network to employ a single router model per class. The corresponding hold tableau is shown in Table 9.

Consider the first and third patterns. It turns out that `class-5` and `class-10` are rare router classes that perform specialized functions in the network and are manufactured by only one vendor. On the other hand, the second and last patterns identify data quality problems. The second pattern reveals that a new router model, namely `Cisco super 5000`, has recently been deployed and was erroneously assigned its own class, with the same class name as the model name. The last pattern captures rows with incomplete information—it turns out that for each row where `class = UNKNOWN`, it is also the case that `model = UNKNOWN` and therefore the above FD is satisfied exactly.

4. RELATED WORK

A great deal of work exists on data quality analysis and data cleaning; see, e.g., [22] for a survey. Many systems have been proposed in this area, including Ajax [11], Bellman [6] and Potter's Wheel [23]. Data Auditor belongs to the class of constraint-driven data quality systems, which also includes SEMANDAQ [8] and StreamClean [18]. SEMANDAQ employs FDs and pattern tableaux to analyze data quality. However, the focus of SEMANDAQ is on identifying violating tuples and suggesting ways to "repair" them. StreamClean proposes a constraint language similar to Data Auditor's, but requires each constraint to hold on the whole relation and focuses on automatic error correction. To the best of our knowledge, Data Auditor is the first system that automatically discovers pattern tableaux to summarize satisfying and violating subsets of the data.

Database integrity constraints have traditionally been used to enforce schema quality. Recently, constraints have been proposed to enforce data quality such as Conditional Functional Dependencies (CFDs) [9], Conditional Inclusion Dependencies (CINDs) [2] and Conditional Sequential Dependencies (CSDs) [13]. 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 summarized by a *pattern tableau*. This concept is also used in Data Auditor.

Many of the constraints used in this paper are closely related to tuple-generating dependencies that assert the existence of certain tuples if some property holds [7]. Here we use them with conditioning to discover tableaux over which such constraints hold (with high or low confidence). Also related are Probabilistic Approximate Constraints (PACs) [20], which were used to detect data quality problems in network traffic databases. For example, a PAC may assert that the inbound and outbound traffic at the two endpoints of a link should be (roughly) equal. However, PACs cannot express all the constraints defined in this paper, especially those with `EXISTS` and `NOT EXISTS` predicates in the `ASSERT` condition.

Finally, we point out the orthogonal problem of discovering which integrity constraints of a certain type hold on a given relation (in contrast to Data Auditor's goal to discover a tableau for a known constraint). Discovering FDs has been studied in [16, 19] discovering CFDs in [3, 10, 24], and discovering Inclusion Dependencies in [5, 21].

5. CONCLUSIONS

In this paper, we presented a case study of pattern-tableau-driven data quality analysis using the Data Auditor tool. Using various types of constraints that detect incorrect, missing and duplicate data, we demonstrated the utility of our approach on a real-life network monitoring database. The pattern tableaux that were generated in our analysis illustrate the extent, and can help determine the root cause, of various types of data quality issues.

We are planning several directions for future work. First, in order to improve efficiency of tableau-driven data quality analysis, we want to study tableau discovery from a small sample of the data (see, e.g., [4] for estimating the confidence of tableaux for FDs from a sample). Second, we plan to extend the space of possible tableau patterns to include conjunctions and negations, similar to those supported by extended CFDs [1].

6. REFERENCES

[1] Loreto Bravo, Wenfei Fan, Floris Geerts, and Shuai Ma. Increasing the Expressivity of Conditional Functional Dependencies without Extra Complexity. *ICDE 2008*, pages 516-525.

[2] Loreto Bravo, Wenfei Fan, and Shuai Ma. Extending dependencies with conditions. *VLDB 2007*, pages 243-254.

[3] Fei Chiang and Renee Miller. Discovering data quality rules. *PVLDB*, 1(1):1166-1177, 2008.

[4] Graham Cormode, Lukasz Golab, Flip Korn, Andrew McGregor, Divesh Srivastava, and Xi Zhang. Estimating the confidence of conditional functional dependencies. *SIGMOD 2009*, pages 469-482.

[5] Olivier Cure. Conditional Inclusion Dependencies for Data Cleansing: Discovery and Violation Detection Issues. *QDB 2009*

[6] Tamraparni Dasu, Theodore Johnson, S. Muthukrishnan and Vladislav Shkapenyuk. Mining database structure; or, how to build a data quality browser. *SIGMOD 2002*, pages 240-251.

[7] Ronald Fagin and Moshe Vardi. The theory of data dependencies - an overview. *ICALP 1984*, pages 1-22.

[8] Wenfei Fan, Floris Geerts and Xibei Jia. Semandaq: A Data Quality System Based on Conditional Functional Dependencies. *PVLDB*, 1(2): 1460-1463, 2008.

[9] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *TODS*, 33(2):1-48, 2008.

[10] Wenfei Fan, Floris Geerts, Laks Lakshmanan and Ming Xiong. Discovering conditional functional dependencies. *ICDE 2009*, pages 1231-1234.

[11] Helena Galhardas, Daniela Florescu, Dennis Shasa, Eric Simon and Cristian-Augustin Saita. Declarative data cleaning: language, model, and algorithms. *VLDB 2001*, pages 371-380.

[12] Lukasz Golab, Theodore Johnson, J. Spencer Seidel, and Vladislav Shkapenyuk. Stream warehousing with DataDepot. *SIGMOD 2009*, pages 847-854.

[13] Lukasz Golab, Howard Karloff, Flip Korn, Avishek Saha, and Divesh Srivastava. Sequential dependencies. *PVLDB* 2(1): 574-585 (2009).

[14] Lukasz Golab, Howard Karloff, Flip Korn, and Divesh Srivastava, Data Auditor: Exploring Data Quality and Semantics using Pattern Tableaux. *PVLDB* 3(2): 1641-1644 (2010).

[15] Lukasz Golab, Howard Karloff, Flip Korn, Divesh Srivastava, and Bei Yu. On generating near-optimal tableaux for conditional functional dependencies. *PVLDB*, 1(1):376-390, 2008.

[16] Yka Huhtala, Juha Karkkainen, Pasi Porkka, and Hanu Toivonen. TANE: An efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 42(2):100-111, 1999.

[17] Theodore Johnson and Damianos Chatziantoniou. Extending complex ad-hoc OLAP. *CIKM 1999*, pages 170-179.

[18] Nodira Khoussainova, Magdalena Balazinska, and Dan Suciu. Towards correcting input data errors probabilistically using integrity constraints. *MobiDE 2006*, pages 43-50.

[19] Jyrki Kivinen and Heikki Mannila. Approximate inference of functional dependencies from relations. *Theoretical Computer Science*, 149(1):129-149, 1995.

[20] Flip Korn, S. Muthukrishnan, and Yunyue Zhu. Checks and balances: Monitoring data quality problems in network traffic databases. *VLDB 2003*, pages 536-547.

[21] Fabien De Marchi, Stephane Lopes, and Jean-Marc Petit. Unary and n-ary inclusion dependency discovery in relational databases. *Journal of Intelligent Information Systems*, 32(1):53–73, 2009.

[22] Erhard Rahm and Hong Hai Do. Data cleaning: problems and current approaches. *IEEE Data Engineering Bulletin*, 23(4): 3-13, 2000.

[23] Vijayshankar Raman and Joseph M. Hellerstein. Potter's wheel: An interactive data cleaning system. *VLDB 2001*, pages 381-390.

[24] Peter Z. Yeh and Colin. A. Puri. Discovering Conditional Functional Dependencies to Detect Data Inconsistencies. *QDB 2010*.