The Data Checking Engine: Complex Rules for Data Quality Monitoring

Felix Heine, Carsten Kleiner, Arne Koschel
University of Applied Sciences & Arts Hannover
Faculty IV, Department of Computer Science, Hannover, Germany
Email: firstname.lastname@hs-hannover.de

Jörg Westermayer
SHS Viveon
Germany
Email: joerg.westermayer@shs-viveon.de

***Abstract*—In the context of data warehousing and business intelligence, data quality is of utmost importance. However, many mid-size data warehouse (DWH) projects do not implement a proper data quality process due to huge up-front investments. Nevertheless, assessing and monitoring data quality is necessary to establish confidence in the DWH data. In this paper, we describe a data quality monitoring system: The "Data Checking Engine" (DCE). The goal of the system is to provide DWH projects with an easy and quickly deployable solution to assess data quality while still providing highest flexibility in the definition of the assessment rules. It allows to express complex quality rules and implements a two-staged template mechanism to facilitate the deployment of large numbers of similar rules. While the rules themselves are SQL statements the tool guides the data quality manager through the process of creating rule templates and rules so that it is rather easy for him to create large sets of quality rules. The rule definition language is illustrated in this paper and we also demonstrate the very flexible capabilities of the DCE by presenting examples of advanced data quality rules and how they can be implemented in the DCE. The usefulness of the DCE has been proven in practical implementations at different clients of SHS Viveon. An impression of the actual implementations of the system is given in terms of the system architecture and GUI screenshots in this paper.**

***Keywords*—Data Quality, Quality Rules, Data Analysis, Data Quality Monitoring, Data Warehouses**

I. Introduction

Data quality (DQ) is of utmost importance for a successful data warehouse project. In this context, continuous monitoring is an integral part of any DQ initiative. In this paper, we describe a data quality monitoring system called *Data Checking Engine* (DCE) developed collaboratively at the University of Applied Sciences & Arts Hannover and SHS Viveon. The main goal is to provide a flexible, yet simple tool to monitor data quality in DWH projects, which can also be used during the DWH development to test its Extract Transform Load (ETL) process. Implementations of the system have already been used at some key pilot customers of SHS Viveon and continuous improvements of the technical as well as the conceptual parts of the system are based on feedback from those customers gathered during daily usage of the system.

Data rules are used in order to constantly monitor the quality of data of a database. For the definition of these rules, a flexible language is necessary. Quality rules are either derived from business rules or found via profiling or data mining. They are executed either in regular intervals or based on specific events like the completion of an ETL job. The results of checking the rules are recorded in a result repository, which also keeps historical data so that users can evaluate the quality of data over time. As rules will evolve over time, it is necessary to keep a history of rule definitions so that historic results can be related to the correct version of the rule's definition.

We believe that the ability to express complex rules is crucial. A set of hard-coded rule types found in some data quality tools is typically only suitable to detect rather simple quality problems on the attribute or single tuple level. However, there are more complex data quality problems, which cannot be detected using such rules. As an example, consider an error located in the logic of an ETL process. Due to this error, the process fails to reference the correct product group for some of the records of a sales fact cube. The bug is subtle and does not show up very often. At the attribute level all sales records are correct. However, the trend of the time series showing the sales sum with respect to individual product groups will indicate a quality problem. Other advanced data quality problems and according check rules will be explained in sec. IV, which is also one of the major extensions of this article in comparison to [1].

It requires skilled users to write such rules, but larger sets of rules will look similar in structure. They differ only in the tables and attributes they are applied to. Therefore, a template mechanism is useful to help users define such rules. The idea is that only the template creator has to cope with the full complexity; template users can then apply these templates to their tables and attributes.

To avoid discontinuity of the reporting environment for DWH users, re-using existing Business Intelligence (BI) tools is superior over building a specialized quality reporting GUI. Still, it is sufficient to export rule results to a quality data mart, which can then be accessed by any standard BI tool. However, the plain rule results have to be aggregated to more comprehensive quality metrics in a flexible and user defined way.

Furthermore, the rules themselves have to be tested in the development environment before deployment. Thus, an automated transfer and synchronization with the production system is necessary.

In a nutshell, we target the following requirements:

- Express complex rules
- Reduce complexity of rules for end users (by utilizing a template mechanism)

- Execute the rules regularly or upon specific events
- Keep a history of rule definitions and execution results
- Store this history in a quality data mart persistently
- Aggregate the rule results to quality metrics
- Provide export/import mechanism for rule meta data

This paper is an extended version of the paper [1]. The example section has been included to describe new quality rule types and to underline the flexibility of our approach, and the related work section has been revised and extended significantly as well.

The remainder of this paper is organized as follows: In the following section, we give an overview of related work. Section III focuses on the definition of quality rules and explains our template mechanism in general, whereas Section IV illustrates the rule definition language in detail by discussing how to implement frequently occurring sample rules. This section is the major extension of this article in comparison to the earlier version [1]. Section V describes the DCE architecture and in the subsequent section we briefly elaborate on quality metrics. Finally, Section VII (which is also an addition in comparison to [1]) illustrates the DCE concept and GUI in more detail, before we summarize our achievements and give an outlook to our future plans in the final section.

II. Related Work

Over the last decade, much research in the data quality domain has been conducted, see for example [2], [3], [4], or [5]. Research areas related to data quality are outlier detection, data deduplication, data quality monitoring, data cleansing, and data mining to detect quality rules. We are specifically interested in monitoring and reporting data quality, and in algorithms to detect quality rules automatically from existing data. In general, we follow the approach of Kimball [6] who outlines an approach to DQ assessment in DWH systems.

For our work, ideas and formalisms to describe quality rules are highly relevant. Many types of quality rules stem from the field of database constraints, as described by Bertossi and Bravo in their survey [7]. As classical constraints are not very flexible, numerous new kinds of constraints have been proposed in the literature. In [3], Fan describes multiple formalisms that target specific quality problems. *Conditional functional dependencies* are used to express more complex rules spanning multiple tuples of a relation (see also [8], [9]), while *conditional inclusion dependencies* are generalizations of referential integrity checks. The classical *edit rules* of Fellegi and Holt are concerned with the integrity of individual records [10]. In [11], Fan defines *editing rules* to match records with master data. Further rule types include *differential dependencies* [12], *multidimensional conditional function dependencies* [13], and *probabilistic, approximate constraints* [14]. From our point of view, these are examples of specific types of rules. We aim to provide a framework that is able to express any of these rules. As all these approaches can be reformulated to SQL, the DCE is able to execute these rules. However, this leads to rather complicated rule definitions. To make life easier for the end users, we further provide a template approach. With this approach, we can define a template for each of the rule types, so that rules can be instantiated in an easy way.

In the domain of data deduplication (also called record linkage), rules are important to describe matching criteria. As an example, the IntelliClean [15] system uses rules like *<if> condition <then> action with probability p* to match duplicates. Fan et al. [16] introduce *matching dependencies* to describe criteria that are used to identify duplicate records. For a survey of duplicate record detection, see [17].

Another approach is to extend SQL to incorporate data quality features. An example is the FraQL [18] language that specifies pivoting features and allows to integrate user defined grouping and aggregate functions that allow to analyze data more comfortably. The drawback is that a special execution engine is required. Thus, the features of existing relational optimizers are not available or have to be reproduced.

Furthermore, many prototypic research systems and commercial tools are present. For an overview, see [19]. Most existing tools focus on dimension data only and thus stress single record problems and deduplication. The profiling component of existing data quality tools currently provides only basic algorithms to detect quality rules, see [20]. We think that more advanced profiling techniques are necessary to detect quality rules automatically. Basic approaches are found in the domains of data mining and outlier detection, also called anomaly detection. An overview can be found in [21] as well as in the recent book of Aggarwal [22]. We plan to integrate these concepts in a later version of the DCE. For this, we are especially interested in finding outliers in time series (see, e.g., [23], [24]) and algorithms to analyze multidimensional data (see, e.g., [25], [26], or [27]). However, to the best of our knowledge, no tool provides a similar mechanism that allows to build complex rule templates, which can, for example, be used to test indicator values against time series models.

III. Rule Definition Language

A central issue is the language to define the quality rules. On the one hand, it has to be expressive to allow complex rules like time series tests. On the other hand, fast definitions of simple rules like NULL value checks has to be possible. Also, the rule execution is typically critical with respect to execution time and resource consumption. As large datasets have to be checked, an efficient rule execution engine is needed.

Thus, we decided to rely on the native SQL executor of the DBMS. This means, the core of each rule is an SQL statement, which collects the required information from the underlying tables. This statement is written by the DCE user, allowing even vendor-specific optimizations like optimizer hints.

DCE defines a *standard attribute set* for the result tuples. The rule statements have to adhere to this standard. Each statement computes a *result value*, which is the basis for the rule check. For a NULL rule, the result value might be the percentage of NULL values of the checked values. There might either be a single result value or multiple values, broken down by dimensional hierarchies. The latter case might for example yield a percentage of NULL values for each product group in each region. Furthermore, two *base values* can be returned. They can provide additional information for the rule outcome. This might be helpful when interpreting the rule results.

For each rule, *multiple bounds* can be defined, specifying valid ranges for the observed values. The bounds can be

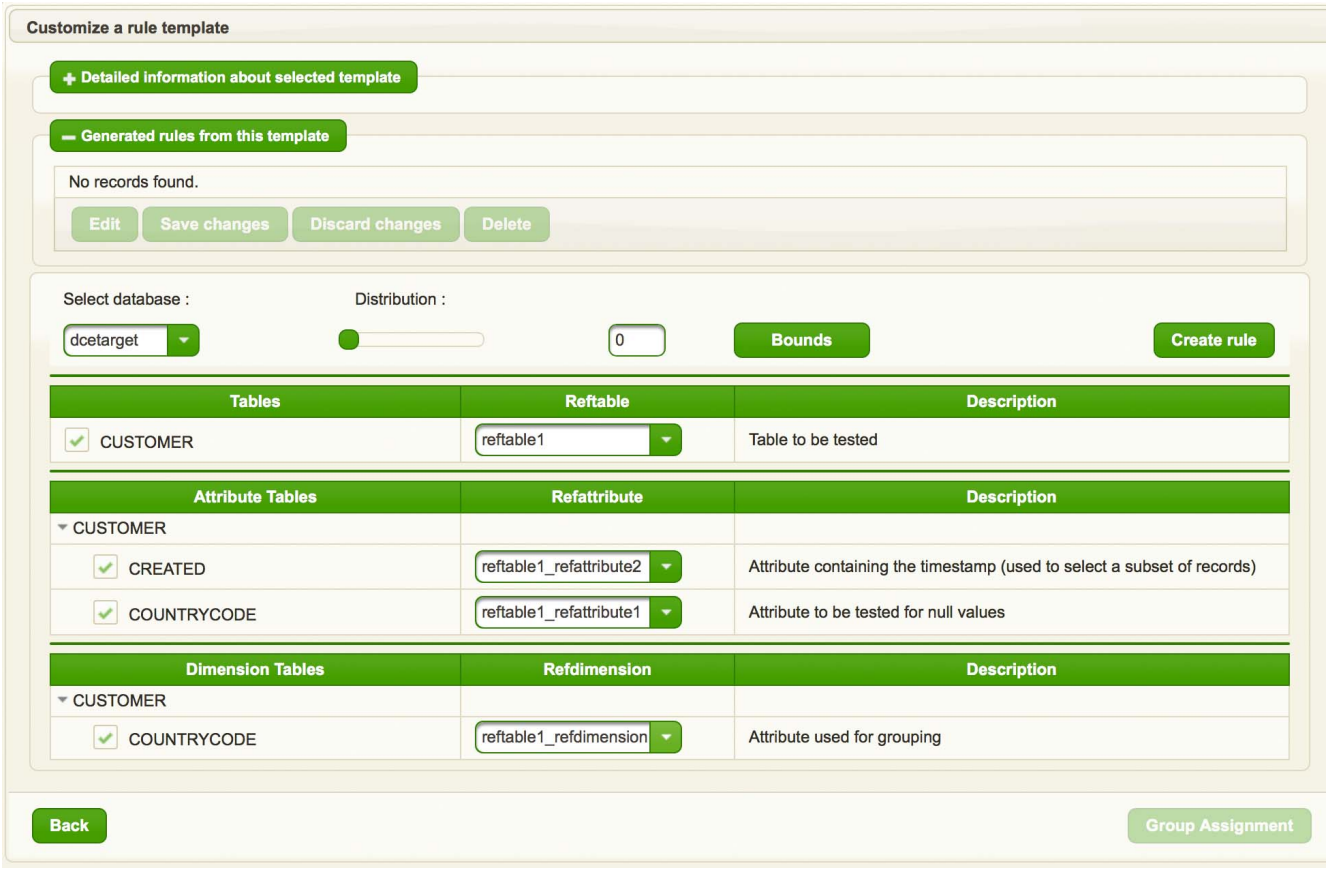

Fig. 1. Instantiating a template

activated or deactivated with respect to all values contained in the result tuple, including both base values and the dimension values. In this way, the bound for NULL values can be normally defined to be 5 percent, however, for specific product groups it might be higher. A specific application for this feature is to change bounds for business metrics, e.g., according to the week day. Typically, the revenue sum for traditional stores might be zero on Sundays.

A *severity* can be assigned to each rule bound, and multiple bounds with different severity can be defined for a rule. The severity information of failed rules is returned to the scheduler. Based on this information, the scheduler might, e.g., decide to interrupt an ETL process or to alert the DWH team.

Each rule's SQL statement can have *multiple parameters*, which are set at execution time. These parameters can for example be used to determine the range of data to be checked. In this way, a quality rule running after an ETL job might be limited to check only the new records in a fact table.

A. Sample rule

In the following, we show how a rule that checks NULL-values does look like. The target is to check the number of NULL value in the middle name in the customer records. As the typical percentage of people that have a middle name varies from country to country, we calculate the values per country. Thus, the country code is used as a dimension value and one result record per country is generated. The two base value fields are used to count the overall number of customers in each country and the number of customers without middle name, respectively. The result value is the percentage of customers in each country without middle name.

```
SELECT
  trunc(sysdate, 'dd') Result_date,
  countrycode dimValues,
  sum(1) baseValue1,
  sum(case when middlename is null
           then 1
           else 0 end) baseValue2,
  round(sum(case when middlename is null
                 then 1 else 0 end) /
        sum(1) * 100) resultValue
FROM customer
WHERE created >
      to_date($date$, 'YYYY-MM-DD')
GROUP BY countrycode
```

Fig. 2. Sample rule code

The SQL for this check is shown in Fig. 2. The result value is then checked against different bounds that are defined on a per-country basis. The rule uses a parameter `$date$` that is used to narrow the check to customers which have been created after the specified date.

```
SELECT
  trunc(sysdate, 'dd') Result_date,
  §reftable1_refdimension1§ dimValues,
  sum(1) baseValue1,
  sum(case when §reftable1_refattribute1§
               is null then 1
          else 0 end) baseValue2,
  round(
   sum(case when §reftable1_refattribute1§
                is null then 1
           else 0 end) /
   sum(1) * 100) resultValue
FROM §reftable1§
WHERE §reftable1_refattribute2§ >
      to_date($date$, 'YYYY-MM-DD')
GROUP BY §reftable1_refdimension1§
```

Fig. 3. Sample template code

B. Templating

In typical environments, there is often a need to define a number of equivalent rules over a large number of tables and attributes. To accommodate for this requirement, we implemented a *template concept*.

A template looks quite similar to a normal rule. It contains an SQL statement producing the same set of standard columns, and it might also contain bound definitions. However, instead of the target table and attribute names, the template's SQL statement contains special markers. For attributes, these markers declare the purpose of the attribute within the rule. Once the user has defined a template, she can instantiate it for multiple sets of tables and attributes. During this process, she either defines new bounds or uses the predefined bounds from the template for the generated rules. The engine forwards rule parameters defined within the template to the generated rules.

C. Example continued

The *sample* statement is a good candidate for a template. In the template, there is another type of parameters called template parameters that are replaced at template instantiation (i.e., rule creation time). These are used to define placeholders for the table and attribute names, like §reftable1§ (cf. Fig. 3).

A *GUI* assists unexperienced users with defining the template parameters, as shown in Fig. 1. In this dialog, the GUI reads the database catalog and lets the user map the template parameters to catalog objects. E.g., §reftable1§ is replaced with sales_fact. Note though that in the current version of the system data type integrity between catalog objects and template parameters is not automatically enforced by the system. I. e. the expert assigning catalog objects to template parameters has to take care that the data types are compatible. If they are not, the execution of the rule will fail with a corresponding SQL error message that will be recorded as a rule execution result in the result repository (cf. Section V). This issue never caused problems in the actual implementations of the system so far, but is a candidate for a future extension, e. g. by providing the opportunity for the template developer to place hints on the expected data type in the template that can assist the person actually creating the rules in the selection process.

IV. Rule Examples

In this section, we illustrate how the rule definition language introduced in the previous section can be used for advanced quality checks. In order to do so, we give examples for quality rules and show how they can be defined within the Data Checking Engine. The section has two purposes: On the one hand, we want to demonstrate that the DCE supports well-known rule types from the data quality literature. On the other hand, we want to introduce new rule types whose capabilities to detect possible quality problems are more sophisticated compared with the well-known rules. We use the AdventureWorks2008DW (in short AWDW) database from Microsoft [28] for our examples.

Data rules are found either by profiling and mining existing data, or by looking at business rules. The mining approach assumes that you either have a data set that is a-priori known to be correct, or you have to do outlier detection and data cleansing on the way. Currently, mining is out of scope for the DCE. This means that we performed the rule mining using external tools (in our case the GNU R tool). However, we plan to integrate this step further with the DCE system in the future, cf. Section VIII.

In general, we distinguish two kinds of rules. Hard rules are those that can clearly identify wrong data. As an example, the violation of a pattern for product codes confirms that the given product code is apparently incorrect. In the same sense, testing a foreign key relationship is a hard constraint. However, there are many quality problems that cannot be detected using hard rules. For this, we need another kind of rule, which we call *value rules*, according to Jack Olsen [5]:

There are additional tests you can construct that point to the presence of inaccurate data that are not as precise in establishing a clear boundary between right and wrong. These are called value rules.

We are going to present some examples of this kind of rules in the last subsections. However, we start with examples for simpler rules that are hard constraints.

A. Pattern for customer alternate key

```
SELECT
  trunc(sysdate, 'dd') Result_date,
  §reftable1_refdimension1§ dimValues,
  0 baseValue1,
  0 baseValue2,
  CASE WHEN
    regexp_like(§reftable1_refattribute1§,
                '$regexp$')
  THEN 1 ELSE 0 END resultValue
FROM §reftable1§
```

Fig. 4. Template for regular expression checking

In AWDW, each customer has an attribute CUSTOMERALTERNATEKEY that contains the business

key consisting of the letters 'A' and 'W' followed by 8 digits. Such a pattern can be validated using a regular expression.

The template shown in Fig. 4 is used for all rules that check regular expressions. To check the alternate key, we instantiate this template using the `dim_customer` table and set the `$regexp$`-parameter to `^AW[0-9]{8}$`.

B. Consistency of translated attributes

```
SELECT
  trunc(sysdate, 'dd') Result_date,
  t1.§reftable1_refdimension1§ dimValues,
  0 baseValue1,
  0 baseValue2,
  count(t2.§reftable1_refdimension1§)
    resultValue
FROM §reftable1§ t1
LEFT JOIN §reftable1§ t2
  ON t1.§reftable1_refattribute1§ =
     t2.§reftable1_refattribute1§
 AND t1.§reftable1_refattribute2§ !=
     t2.§reftable1_refattribute2§
GROUP BY t1.§reftable1_refdimension1§
```

Fig. 5. Template for functional dependency checking

In the AWDW dimension tables, some descriptions are present in multiple languages. As the same English text is expected to have always the same translation, a functional dependency (FD) is present. To check FDs, we have written a template (see Fig. 5). The template searches for tuple combinations that agree upon the first attribute's value and disagree upon the second attribute. For each tuple, the number of tuples that have a different value in the second attribute is counted.

We instantiated the template, e.g., for the attributes `ENGLISHEDUCATION` and `SPANISHEDUCATION` in the `dim_customer`-table. For each tuple in the table, the rule counts the number of tuples that match the current tuple with respect to `ENGLISHEDUCATION` but disagree in `SPANISHEDUCATION`. We have modified the text for `SPANISHEDUCATION` in a single tuple. As there are 5099 tuples that share the same text in `ENGLISHEDUCATION`, each of these tuples gets a resultValue of 1, while the modified tuple gets a value of 5098. Although the probability is high that the translation is correct for 5099 tuples and it is wrong for only a single tuple, the rule cannot make this distinction. Thus, the bound for the rule is zero, meaning each of the tuples is regarded to be a potential quality error.

C. Sales count of clothing and accessories

Now we start with examples for value rules. First, we are going to check whether the sales count (items per day) of the two product categories clothing and accessories is reasonable. A scatter plot of the counts per day in Fig. 6 indicates that there is a correlation between the two variables. Indeed, the correlation coefficient is 0.7469.

We are going to exploit this to build a quality rule. The idea is to estimate a bivariate normal distribution from the data

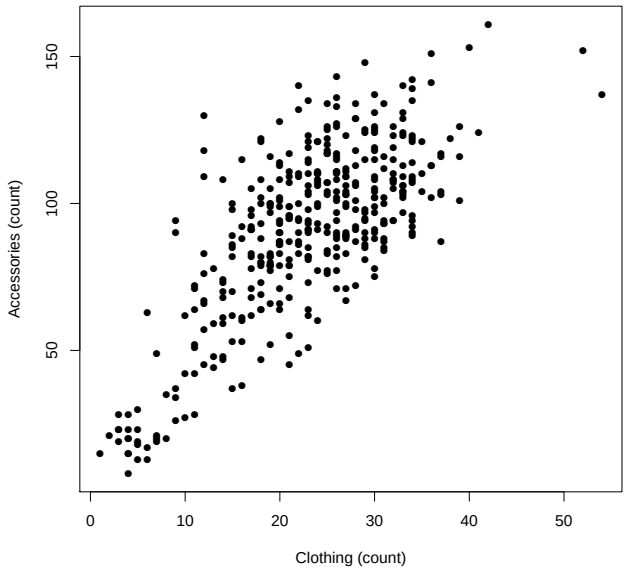

Fig. 6. Item sales per day in two categories

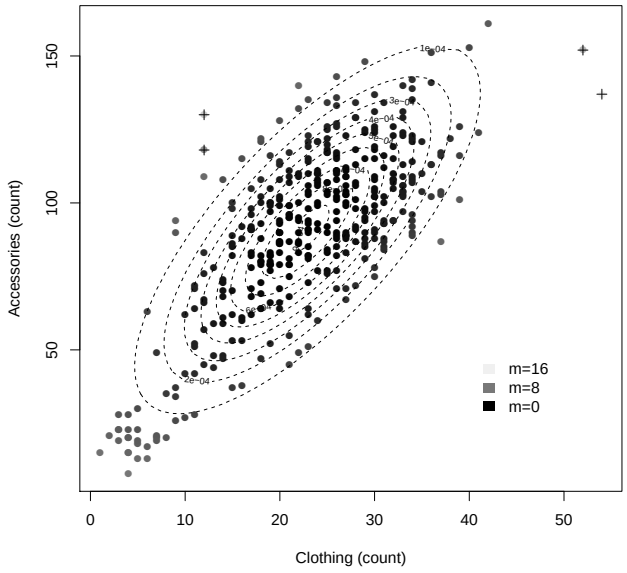

Fig. 7. Estimated distribution and distance

and to use the squared Mahalanobis distance from the center of the distribution as an outlier score for each data point. The estimation uses the variance-covariance matrix S of the data as an estimate for the covariance and the sample mean of the data $\bar{x}$ as the mean of the distribution. The contour lines of the density of the resulting distribution are displayed in Fig. 7.

The squared Mahalanobis distance is calculated using the following equation (see [29, p. 662]). Note that we assume

that x is a column vector.

$$m^2 = (x - \bar{x})^T S^{-1} (x - \bar{x}) \quad (1)$$

The distance values are shown in the figure using a grey scale. An important property of the Mahalanobis distance is that it takes the correlation between the attributes into account. As an example, the point $(40, 153)$ is not an outlier, while the point $(12, 118)$ is an outlier although it is more close to the center in terms of a Euclidean distance. This is because $(40, 153)$ better fits with the distribution of the data.

```
SELECT trunc(sysdate, 'dd') Result_date,
       key dimValues,
       x1 baseValue1,
       x2 baseValue2,
       msq resultValue
FROM
(SELECT (xn1*(xn1*$si11$ + xn2*$si21$) +
         xn2*(xn1*$si12$ + xn2*$si22$))
        msq,
        x1, x2, key
 FROM
 (SELECT x1-$xm1$ as xn1,
         x2-$xm2$ as xn2, x1, x2, key
  FROM
  (SELECT §reftable1_refattribute1§ x1,
          §reftable1_refattribute2§ x2,
          §reftable1_refdimension1§ key
   FROM §reftable1§
  )
 )
)
```

Fig. 8. Template to calculate Mahalanobis distances

The rule template in Fig. 8 calculates the squared Mahalanobis distance. It is based on a view `c34` that aggregates the sales count in both categories on a daily basis. The entries of S^{-1} are provided as parameters `$si11$`, etc. and $\bar{x}$ is provided using the parameters `$xm1$` and `$xm2$`.

We instantiated the template and used a bound of 9 on the distance. So each data point with a distance greater than 9 is declared to be suspicious. The corresponding points which are detected as errors by the rule are marked with a plus in Fig. 7. Please note that these errors have to be interpreted differently compared to the previous errors. Errors reported by value rules are unusual values that have to be investigated further by domain experts but do not immediately mean that a data quality issue has been detected.

D. Age distribution of customers

As another example for a value rule, we want to check whether the ages of our customers are reasonable. We assume that the current age distribution is correct and use it as a reference distribution. Our goal is to develop a quality rule that generates a warning when the distribution changes significantly. To achieve this, we start by creating a view that displays the current age distribution. The view is called `customer_age`. The distribution initially looks as shown in Fig. 9 (light gray bars labeled "original"). Note that in fact the distribution shows that AdventureWork's customers

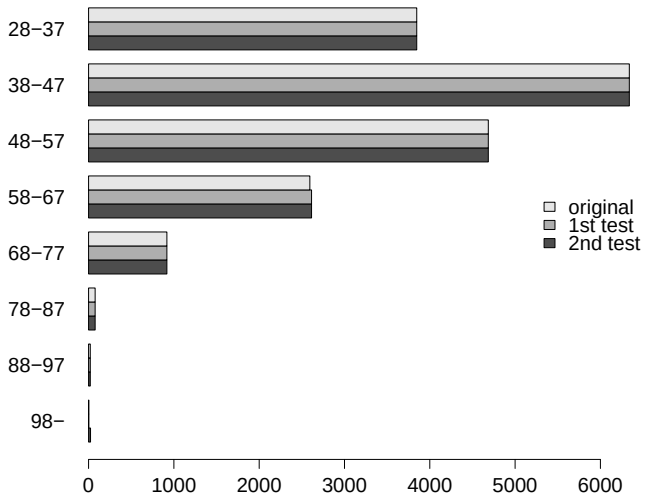

Fig. 9. Original age distribution and test modifications

```
SELECT trunc(sysdate, 'dd') result_date,
       1 dimValues,
       0 baseValue1,
       0 baseValue2,
       sum((N1-E1)*(N1-E1)/E1 +
           (N2-E2)*(N2-E2)/E2) resultValue
FROM (
SELECT nvl(N1, 0) N1, nvl(N2, 0) N2,
       N1SUM * (nvl(N1, 0) + nvl(N2, 0))
               / (N1SUM+N2SUM) E1,
       N2SUM * (nvl(N1, 0) + nvl(N2, 0))
               / (N1SUM+N2SUM) E2
FROM
(SELECT §reftable1_refattribute1§ key,
        §reftable1_refattribute2§ N1
 FROM §reftable1§) D1
FULL OUTER JOIN
(SELECT §reftable2_refattribute1§ key,
        §reftable2_refattribute2§ N2
 FROM §reftable2§) D2
ON (D1.key = D2.key),
(SELECT sum(§reftable1_refattribute2§)
        N1SUM FROM §reftable1§) N1,
(SELECT sum(§reftable2_refattribute2§)
        N2SUM FROM §reftable2§) N2
)
```

Fig. 10. Template for χ^2 homogeneity test

are quite old, which might already indicate a data quality problem. However, we ignore this for the sake of the example and assume that the initial distribution is correct.

We now store a snapshot of the view in the table `customer_age_ref`. This table will serve as a reference for the current age distribution. Now we can use a χ^2 homogeneity test to see whether the current customer's ages are similarly

distributed. The test statistic is defined as follows:

$$\chi^2 = \sum_{j=1}^{k}\sum_{i=1}^{m}\frac{(n_{ij} - E_{ij})^2}{E_{ij}} \quad (2)$$

In this formula, k is the number of samples to be compared. As we have two samples (the reference age distribution vs. the current distribution), we have $k = 2$. Furthermore, m is the number of groups, in our case $m = 8$ as we have eight age groups. n_{ij} is the number of customers in sample j belonging to age group i. E_{ij} is the expected number of customers in sample j in age group i. It is calculated using the margins:

$$E_{ij} = \frac{n_{i\cdot}n_{\cdot j}}{n} \quad (3)$$

Here, $n_{i\cdot}$ is the overall number of items in group i in both samples and $n_{\cdot j}$ is the size of sample j.

Again, we define a rule template to calculate the statistic. It is shown in Fig. 10.

The result of this statement is the test statistic for the χ^2 test. Its value has to be compared with an appropriate quantile of the χ^2 distribution: $\chi^2_{(k-1)(m-1);0.95} = \chi^2_{7;0.95} = 14.067$. We use this value as an upper bound. All values above this one will generate a quality error.

When we initially run the rule, the output is (as expected) 0. This is due to the fact that we compare two identical samples. To test the rule, we first insert 20 customers of age 59. They are in age group 58-67, which already has nearly 2600 customers. The rule now yields a value of 0.066, which is way below our bound, as expected. Now we start to insert more unusual customers of age 99. As we insert 5 of them, the value already becomes 2.33. With 15 more customers of this age, we reach 15.418, which generates a quality error.

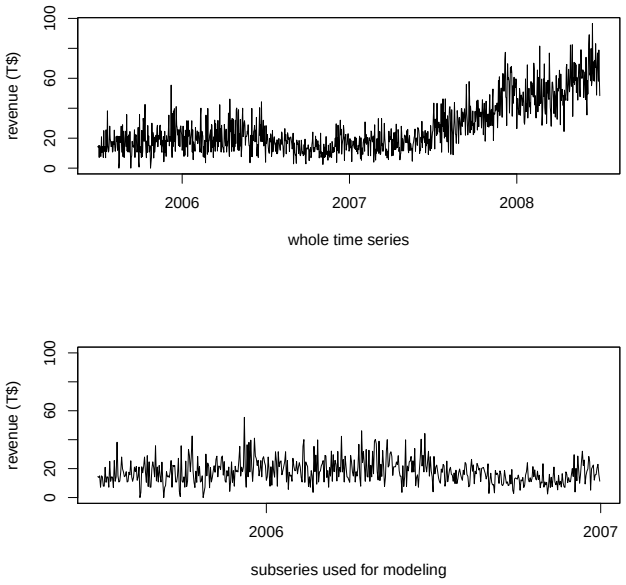

Fig. 11. Daily internet revenue

E. Check the revenue time series

In our final example, we check the overall daily Internet revenue of the AdventureWorks company. It can be calculated by aggregating the facts in `fact_internet_sales`. Again, the underlying idea of the rule is to define a model for the data and to check that the new data does not contradict the model significantly. For this example, we use the data up to the end of 2006 to define the model and then check the data from 2007 on against the model. Fig. 11 shows the complete time series and an enlarged plot of the part used for modelling.

In this case, a time series model is appropriate and we use the ARIMA family of models [30]. These are stochastic models that are composed of an auto-regessive part (AR), an integration part (I), and a moving average part (MA). When the integration part is present, the differenced time series is used instead of the original one. The basis for the series is a series of random components. They are assumed to be independent and normally distributed. The AR part captures dependencies between the current value and the preceding values, while the MA part is used to model a moving average of the past random components.

A first look at the plots indicates that the series is non-stationary, as there seems to be a trend, at least from mid-2007 on. The variance looks fairly large and there is no sign of seasonality.

We use the `auto.arima` function of R to select an appropriate ARIMA model and to estimate the parameters. The result is an ARIMA(0,1,2) model. This means that no AR part is present, the original series is differentiated once, and the MA part averages over the past two random components. The following coefficients are estimated:

$$y_t - y_{t-1} = e_t + -1.0012e_{t-1} + 0.0626e_{t-2} \quad (4)$$

The random components are denoted by e_t. We can see that the negative last random value has a huge influence on the current observation (coefficient -1.0012). The estimated standard deviation is $\hat{s} = 7951.597$. The AIC value is rather large (11407).

The basic idea of the rule is to use the model to calculate each day a one step forecast and to check whether the newly observed value is within the forecast interval. If it is outside, then we generate an error, as the value is rather unlikely. For the forecast, we use equation (4) and set $e_t = 0$, as we know nothing about the current random component. The past two values e_{t-1} and e_{t-2} are estimated using the past residuals (i.e., the differences between the past forecasts and observed values). We denote the residuals with $\hat{e}_{t-1}$ and $\hat{e}_{t-2}$. This yields the following equation for the point forecast $\hat{y}_t$:

$$\hat{y}_t = y_{t-1} + -1.0012\hat{e}_{t-1} + 0.0626\hat{e}_{t-2} \quad (5)$$

The 95% forecast interval can be computed as $\hat{y}_t \pm \hat{s} * 1.96$.

In order to build a DCE rule using this model we have to store the past residuals so that we can access them during the current rule execution. We use the DCE repository's rule outcome table for this task. In this way, each day's run stores the current residual in the repository. We have to bootstrap the rule by feeding at least two residuals in the results table so that the rule can start to calculate the next forecasts. These

past residuals are included in the R object generated by the `auto.arima` function. Currently, we have to store them in the repository by hand. Later, we plan to automate this task in a profiling phase.

```
SELECT
  trunc(sysdate, 'dd') Result_date,
  datekey dimValues,
  extendedamount baseValue1,
  0 baseValue2,
  get_residual('$checkdate$') resultValue
FROM daily_revenue
WHERE fulldatealternatekey =
      to_date('$checkdate$', 'YYYY-MM-DD')
```

Fig. 12. Rule to check a time series

We implemented the forecast equation (5) as a stored function `get_residual` within the database. The function only returns the residual, as this is all we need to check the rule bounds and to prepare for the next run of the rule. Using this stored function, the rule's SQL is quite simple, as shown in Fig. 12. The view `daily_revenue` calculates daily aggregates of the internet sales revenue.

The rule uses a parameter `$checkdate$` that specifies the target date. Each rule run checks a single day. As the rule depends on the past two runs we have to ensure that the rule runs every day without omissions.

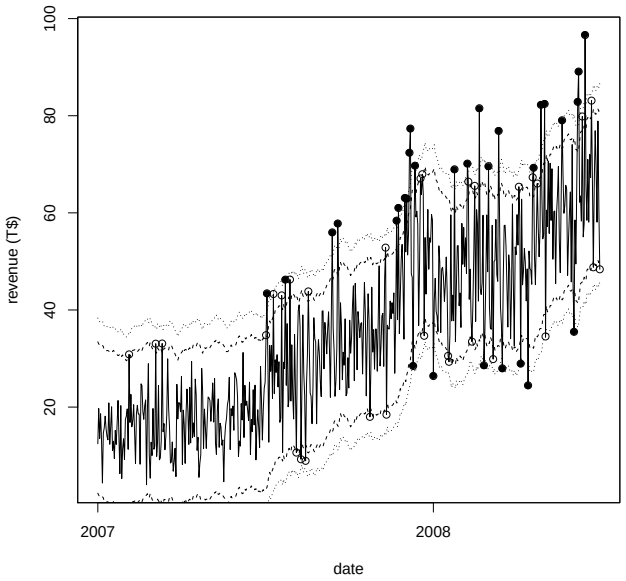

Fig. 13. Prediction intervals and warned values

Fig. 13 shows the second part of the time series and the forecast intervals calculated from the above formula for 95% (dashed) and 99% (dotted). Circles indicate data points that are outside the 95%; filled circles indicate those that are also outside the 99% interval. Depending on the bounds specified in the rule, the DCE reports these values.

V. ARCHITECTURE

In this section, we will illustrate the system architecture of the current DCE implementation, which is capable of checking the previously explained data quality rules. We also show how the DCE fits into a typical enterprise DWH implementation. Fig. 14 shows an overview of the DCE overall architecture. The DCE itself is organized as a classical three-tier application. It interacts with the enterprise data warehouse system in order to compute quality indicators. Also, results of the data quality checks may be propagated into another external database system, the data quality data mart. This database in itself is also organized as a data mart and provides long term storage of computed data quality indicators in order to be used for long term analysis of enterprise wide data quality. In a sense it is a meta-data warehouse for data quality. There is also an external scheduling component (typically standard system scheduling capabilities), which triggers computation of data quality indicators at previously defined points in time.

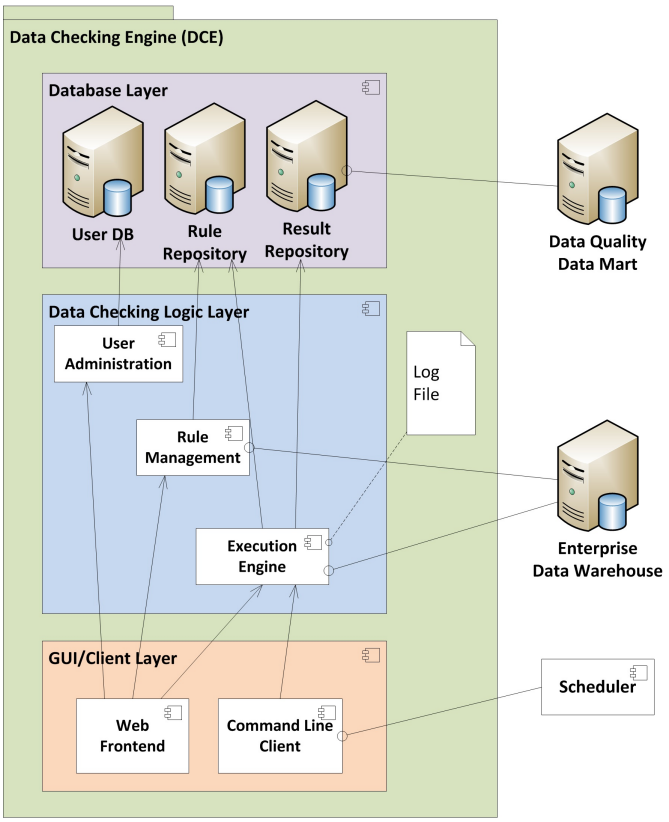

Fig. 14. Data checking engine architecture overview

Within the DCE itself the main entry point for data quality managers is the GUI of the DCE web application (shown at the bottom of Fig. 14). The GUI is used to manage users of the DCE application, to manage data quality rules, and to manage data rule executions. As typically the execution of data quality checks is not triggered manually, there is also a command-line client library for the rule execution engine that is triggered by an external scheduler. The schedule to be used is managed in the web application as well.

The main data checking business logic can be found in the middle tier. This logic is used by the web application

as described above. Note that there is a strict separation between user management, rule management and rule execution management in the middle tier as well. Whereas the user administration component provides standard functionality, note that the rule management component contains advanced features. For instance the template mechanism described in the previous section is implemented here.

The execution engine is also managed by the web application: on the one hand, rules can be manually executed from the web application, on the other hand, scheduled execution can be defined here.

During rule execution, the engine replaces the parameters in the rule's SQL statement with their current values and then runs the statement using the target database. Thus, moving large amounts of data into the DCE engine is avoided. The result of the SQL statement is then further processed. This includes checking the currently applicable bounds and testing their severity.

In the execution engine, it is also defined, which rules are executed on what data warehouse database under whose privileges. Note that multiple different data warehouses (or database systems) may be used as source, because the connection information is also managed by the web application.

Note that the performance of the engine itself is not critical. All rules are finally translated to SQL and executed on the target database. Compared to the execution time of the SQL statement (which perhaps runs on large data sets), the preparation phase and the interpretation of the results, which includes bound checking, is negligible. Thus, the scalability of the DCE depends heavily on the scalability of the target database. In case of performance problems, typical database tuning options like indexing or materialization of views are used.

Finally, the database layer consists of three separate areas:

- Rule repository, which holds the data quality rules as well as base templates
- Result repository holding results of rule execution
- User database which is used for access management to only the DCE itself

Once results of the executed data quality rules have been stored in the result repository they may be propagated to the data quality data mart that aggregates the results into quality indicators.

This data mart is not part of the DCE but located within the standard DWH infrastructure of the company. Thus, standard interfaces such as reporting and BI tools can be used to further present and analyze the data quality status. This way the additional effort for data quality monitoring can be kept minimal as access to data quality indicators follows well established processes and uses well-known tools, which are used for regular monitoring of enterprise performance indicators as well. In addition, the concept of viewing data quality indicators similar to regular performance indicators is very fitting, as these have to be tracked accordingly in order to ensure reliability of data in the data warehouse. Ultimately, this is necessary to make the right entrepreneurial decisions based on reliable information.

VI. Data Quality Metrics

The result repository contains large amounts of specific results that individually describe only a very small fraction of the overall data quality of the DWH. In order to get a quick overview of the quality level, a small set of metrics that aggregate the rule results is required.

In the literature, there are various approaches to define data quality indicators, for example [31]. Thus, we decided to provide a flexible approach that enables the user to define her own indicator hierarchies. The engine stores indicator definition meta data and calculates the resulting indicator values.

An important issue here is to take incremental checks into account. As an example, consider a rule that checks the number of dimension foreign keys in a fact table that reference a dummy instead of a real dimension entry. As the fact table is large, the daily rule just checks the new fact records loaded in the previous ETL run. Thus, the indicator has to aggregate over the current and past runs to provide an overall view of the completeness of the dimension values.

VII. DCE 'Look and Feel' Impressions

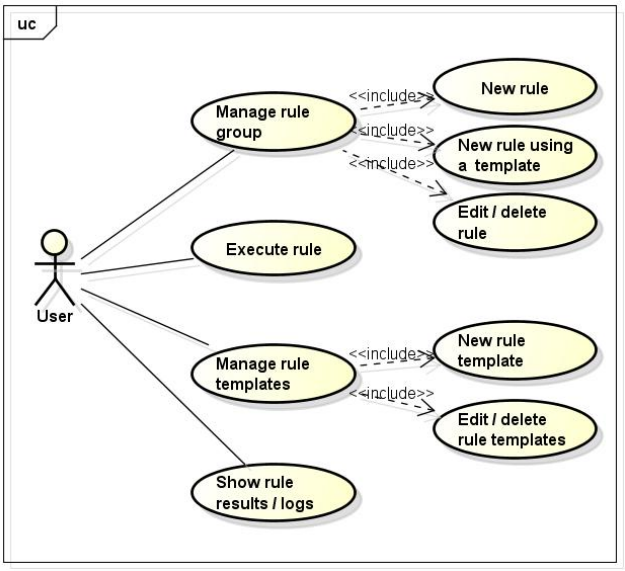

Fig. 15. End user – Major use cases

Based on the above discussed concepts and architecture we implemented the overall DCE system. To give an impression how DCE actually looks like from a usage perspective, we will illustrate in this section some selected use cases and screen shots from the DCE GUI.

A. Use Cases

We can distinguish between two main types of users within DCE, namely end users and administrators.

End users are the actual rule authors. Based on their domain knowledge, they formulate DCE rules as described above. Respectively, their main use cases (after their login to DCE) are 'show rule group', 'input rules' maybe based on an existing

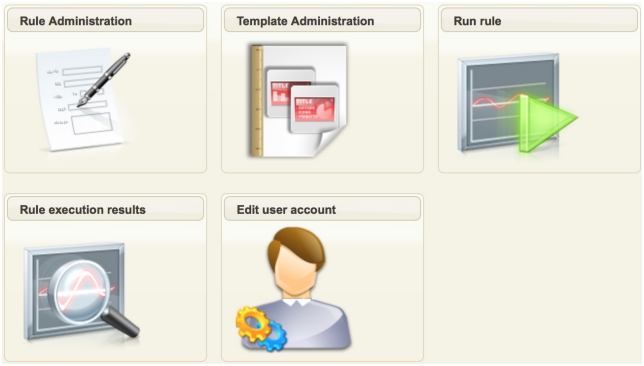

Fig. 16. User's central menu

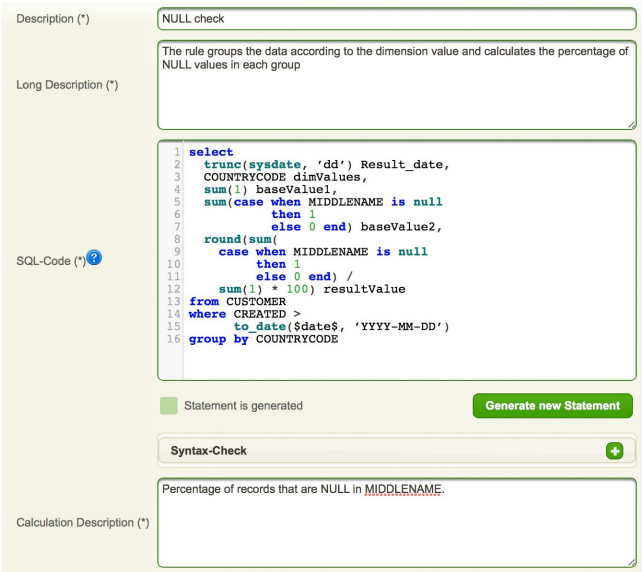

Fig. 17. User's rule insert / edit screen

one, 'edit rule', and 'show rule results'. As explained, rules could be based on rule templates. The main use cases are illustrated in Fig. 15.

Administrators have as their main responsibilities the management of user accounts, database connections, and general settings.

Name		Start Date	End Date	Parameters	Value	Comparison	Result
NULL check (Group ID: 301)		2014-02-27 09:12:13.952	2014-02-27 09:12:14.078				Passed
NULL check (Group ID: 301)		2014-02-27 09:14:57.832	2014-02-27 09:14:57.953				Passed
NULL check (Rule ID: 401)		2014-02-27 09:14:57.87	2014-02-27 09:14:57.943	date: 2010-01-01			Passed
AU					42.0	<= 50.0; 0.0	Passed (0)
CA					42.0	<= 50.0; 0.0	Passed (0)
DE					42.0	<= 70.0; 0.0	Passed (0)
FR					42.0	<= 70.0; 0.0	Passed (0)
GB					43.0	<= 50.0; 0.0	Passed (0)
US					42.0	<= 50.0; 0.0	Passed (0)

Fig. 18. Execution results 1

Name	Description	Start Date	End Date		Comparison	Result
agetest (Group ID: 153)		2014-01-29 18:37:56.572	2014-01-29 18:37:56.835			Passed
agetest (Group ID: 153)		2014-01-29 19:36:50.199	2014-01-29 19:36:50.507			Passed
agetest (Group ID: 153)		2014-01-29 19:37:15.881	2014-01-29 19:37:16.561			Passed
homogeneity test (Rule ID: 205)	uses chi squared	2014-01-29 19:37:16.298	2014-01-29 19:37:16.556			Passed
1				0.065	< 14.06714; 0.0	Passed (0)
agetest (Group ID: 153)		2014-01-29 19:37:52.25	2014-01-29 19:37:52.982			Passed
homogeneity test (Rule ID: 205)	uses chi squared	2014-01-29 19:37:52.704	2014-01-29 19:37:52.976			Passed
1				2.332	< 14.06714; 0.0	Passed (0)
agetest (Group ID: 153)		2014-01-29 19:38:55.73	2014-01-29 19:38:56.057			Error
homogeneity test (Rule ID: 205)	uses chi squared	2014-01-29 19:38:55.778	2014-01-29 19:38:56.052			Error (0)
1				15.41	< 14.06714; 0.0	Error (0)

Fig. 19. Execution results 2

B. Screen Shots

To give an impression of the 'look and feel' of the DCE GUI, we show a few screen shots here.

Fig. 16 shows the central menu options for a DCE end user, namely rule and template management, rule execution, execution results inspection, and edit user account.

A typical DCE end user task is the insertion of new rules. Fig. 17 shows the corresponding screen, which allows a DCE end user to edit new rules.

Two typical result screens from rule execution are shown in Fig. 18 and Fig. 19. In Fig. 19 results from failed rule checks (errors) are shown as well.

In addition, further options exist, e. g., to edit rule templates, to group rules, and options for administrators as well.

VIII. Conclusion and Future Work

In this paper, we have presented a motivation why complex data quality rules are required in most practical enterprise application scenarios. We have developed a rule definition language with a two-stage templating mechanism in order to be able to express most of these rules for SQL-based data sources. The language along with the templating mechanism is both powerful enough to express complex rules as well as simple enough to be used by domain experts in practice. This has been proven by a prototypical implementation based on the described architecture. This tool has been validated by interviewing teams of different DWH projects and building project specific prototypical setups in a real world environment. Our engine has been able to support their quality monitoring requirements. Especially, the flexibility in rule definition was appreciated. We have not only detected quality problems on tuple level but also more complex issues, e.g., checking the trend of indicators stored in a fact table. As expected, our template mechanism has proven to be an important way to simplify rule definition.

The engine keeps a comprehensive history of rule results *and* rule meta data, which allows to monitor data quality over time and to check whether quality improvement projects were successful. This quality data is exposed to external BI tools for

reporting and further analysis. This integration of quality data into tools for further analysis will be significantly improved and simplified in the future. We aim to provide semi-automated aggregation features that provide the data quality manager in an enterprise with a set of traffic lights on a dashboard-like portal in order to get an impression of the overall data quality quickly and easily.

An important consequence of the flexibility of our approach is that the DCE can also be used during DWH/ETL development to test the result processes. The testing rules developed during this project phase may also be used during normal operation later on, reducing the overall cost of data quality. Practical experiences with this kind of dual use of rules will be gathered in the future. An advantage of this approach might be an improved quality of the quality rules themselves as their definition will be based on thoroughly developed concepts.

Our approach is currently working on any relational database system. In the future, we plan to also integrate data in other data sources such as Big Data systems like Hadoop and other NoSQL database systems, as more and more relevant data will be stored there. Thus, data quality should be monitored there as well. As there is currently no universal query language standard like SQL in the relational sector, we will have to devise a flexible way to cope with various rule definition languages and/or define a generic rule definition language together with automated translations into the languages of the source systems.

We will also work further on the process of determining quality rules. For newly defined rules the data quality manager faces the issue that it is difficult to determine whether there is an actual data quality problem or an inaccuracy in the rule. Consequently, we plan to further explore the possibility to semi-automatically derive data quality rules by advanced data profiling methods. These may even be enhanced by integrating data mining processes.

Finally, there is still a need for even more advanced data quality rules that are based on more sophisticated statistical models (both static and dynamic) as well as multi-variate time series data. While basic ideas for this have been presented in Section IV, in the future we would like to extend this approach and also reduce the manual effort by semi-automated steps.

REFERENCES

[1] F. Heine, C. Kleiner, A. Koschel, and J. Westermayer, "The data checking engine: Monitoring data quality," in *DATA ANALYTICS 2013, The Second International Conference on Data Analytics*, 2013, pp. 7–10.

[2] S. Sadiq, Ed., *Handbook of Data Quality*, 1st ed. Springer, 2013.

[3] W. Fan and F. Geerts, *Foundations of Data Quality Management*, ser. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.

[4] C. Batini and M. Scannapieco, *Data Quality: Concepts, Methodologies and Techniques*, 1st ed. Springer, 2006.

[5] J. Olson, *Data Quality. The Accuracy Dimension*. Morgan Kaufmann, 2002.

[6] R. Kimball and J. Caserta, *The data warehouse ETL toolkit*. Wiley, 2004.

[7] L. Bertossi and L. Bravo, *Handbook of Data Quality*, 1st ed. Springer, 2013, ch. Generic and Declarative Approaches to Data Quality Management, pp. 181–211.

[8] P. Z. Yeh and C. A. Puri, "An efficient and robust approach for discovering data quality rules," in *22nd International Conference on Tools with Artificial Intelligence*, 2010.

[9] F. Chiang and R. J. Miller, "Discovering data quality rules," in *Proceedings of the VLDB 08*, 2008.

[10] I. P. Fellegi and D. Holt, "A systematic approach to automatic edit and imputation," *Journal of the American Statistical Association*, vol. 71, no. 353, pp. 17–35, 1976.

[11] W. Fan, J. Li, S. Ma, N. Tang, and W. Yu, "Towards certain fixes with editing rules and master data," *The VLDB Journal*, vol. 21, no. 2, pp. 213–238, Apr. 2012. [Online]. Available: http://dx.doi.org/10.1007/s00778-011-0253-7

[12] S. Song and L. Chen, "Differential dependencies: Reasoning and discovery," *ACM Transactions on Database Systems*, vol. 36, no. 3, pp. 16:1–16:41, Aug. 2011. [Online]. Available: http://doi.acm.org/10.1145/2000824.2000826

[13] S. Brüggemann, "Addressing internal consistency with multidimensional conditional functional dependencies," in *International Conference on Management of Data COMAD 2010, Nagpur, India*, 2010.

[14] F. Korn, S. Muthukrishnan, and Y. Zhu, "Checks and balances: Monitoring data quality problems in network traffic databases," in *Proceedings of the 29th VLDB Conference, Berlin*, 2003.

[15] M. L. Lee, T. W. Ling, and W. L. Low, "Intelliclean: A knowledge-based intelligent data cleaner," in *ACM SIGKDD, Boston, 2000*, 2000.

[16] W. Fan, H. Gao, X. Jia, J. Li, and S. Ma, "Dynamic constraints for record matching," *The VLDB Journal*, vol. 20, pp. 495–520, 2011.

[17] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate record detection: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, 2007.

[18] K. Sattler, S. Conrad, and G. Saake, "Adding conflict resolution features to a query language for database federations," in *Proc. 3nd Int. Workshop on Engineering Federated Information Systems, EFIS'00, Dublin, Ireland, June*, 2000, pp. 41–52.

[19] J. Barateiro and H. Galhardas, "A survey of data quality tools," *Datenbank-Spektrum*, vol. 14, 2005.

[20] F. Naumann, "Data profiling revisited," *SIGMOD Record*, 2013.

[21] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 15:1–15:58, Jul. 2009. [Online]. Available: http://doi.acm.org/10.1145/1541880.1541882

[22] C. C. Aggarwal, *Outlier Analysis*. Springer, 2013.

[23] K. Yamanishi and J. Takeuchi, "A unifying framework for detecting outliers and change points from non-stationary time series data," in *Proceedings of the Eighth ACM SIGKDD-02*, 2002.

[24] C. S. Hilas, I. T. Rekanos, S. K. Goudos, P. A. Mastorocostas, and J. N. Sahalos, "Level change detection in time series using higher order statistics," in *16th International Conference on Digital Signal Processing*, 2009.

[25] S. Sarawagi, R. Agrawal, and N. Megiddo, "Discovery-driven exploration of olap data cubes," in *Advances in Database Technology — EDBT'98*, 1998.

[26] E. Müller, M. Schiffer, and T. Seidl, "Statistical selection of relevant subspace projections for outlier ranking," in *27th IEEE International Conference on Data Engineering (ICDE)*, 2011.

[27] C. Ordonez and Z. Chen, "Evaluating statistical tests on olap cubes to compare degree of disease," *IEEE Transactions on Information Technology in Biomedicine*, vol. 13, no. 5, 2009.

[28] Microsoft. (2014, Feb) Microsoft sql server database product samples. [Online]. Available: http://msftdbprodsamples.codeplex.com

[29] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Pearson, 2006.

[30] A. C. Harvey, *Time Series Models*. Pearson Education, 1993.

[31] B. Heinrich, M. Kaiser, and M. Klier, "Metrics for measuring data quality - foundations for an economic oriented management of data quality," in *Proceedings of the 2nd International Conference on Software and Data Technologies (ICSOFT). INSTICC/Polytechnic Institute of Setúbal*, J. Filipe, B. Shishkov, and M. Helfert, Eds., 2007.