Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 180 (2021) 772–777

Procedia Computer Science

www.elsevier.com/locate/procedia

International Conference on Industry 4.0 and Smart Manufacturing

DaQL 2.0: Measure Data Quality based on Entity Models

Christian Lettner[a], Reinhard Stumptner[b], Werner Fragner[c], Franz Rauchenzauner[d], Lisa Ehrlinger[a,e]

[a] *Software Competence Center Hagenberg GmbH (SCCH), Softwarepark 21, 4232 Hagenberg, Austria*
[b] *FAW GmbH, Softwarepark 35, 4232 Hagenberg, Austria*
[c] *STIWA Automation GmbH, Softwarepark 37, 4232 Hagenberg, Austria*
[d] *STIWA Automation GmbH, Salzburgerstraße 52, 4800 Attnang-Puchheim, Austria*
[e] *Johannes Kepler University Linz, Altenberger Straße 69, 4040 Linz, Austria*

Abstract

In order to make good decisions, the data used for decision-making needs to be of high quality. As the volume of data continually increases, ensuring high data quality is a big challenge nowadays and needs to be automated with tools. The goal of the Data Quality Library (DaQL) is to provide a tool to continuously ensure and measure data quality as proposed in [5]. In this paper, we present the current status of the development of the new DaQL version 2.0. The main contribution of DaQL 2.0 is the possibility to define data quality rules for complex data objects (called entities), which represent business objects. In contrast to existing tools, a user does not require detailed knowledge about the database schema that is observed.

Keywords: Data Quality; Domain Specific Language; Data Quality Library; DaQL

1. Introduction

It has been known for years, poor data quality leads to problems like mistakes in operational processes (e.g., manufacturing, sales), wrong measures in analytical applications, which may lead to wrong decisions, frustration of customers, or business partners and general mistrust [12]. Data is often "dirty" and this issue costs "trillions of dollars" [10] per year. The following questions play a major role in connection with data quality. Is the data complete and correctly used in the applications? Are data values consistent throughout an entire organization? Do the data values match certain properties? Are the data values accurate? Were any data values duplicated? Although the problem and its

* Corresponding author. Tel.: + 43 50343-837.
E-mail address: christian.lettner@scch.at

consequences have been present for a long time, data quality management is still poorly implemented in practice and it is a big challenge to overcome these problems. But we think that with supportive tools for measuring and managing data quality (rule definition, continuous monitoring etc.) many institutions can be encouraged to actively tackle their data quality issues. DaQL (Data Quality Library), originally presented in [5], is designed to continuously monitor data quality, based on the four requirements for automated data quality monitoring defined in [7]. The main enhancement of DaQL 2.0 is the possibility to define data quality rules for complex data objects or data entities, which means that a rule may not only consider a single data object but it can also consider its relations and related objects. Consequently it is possible to define rules, which check data including their relations in a very compact manner.

2. Related Work

This section provides an overview on alternative approaches and tools to measure data quality. Cichy and Rass [3] provide a recent survey on data quality frameworks and Ehrlinger et al. [6] a survey on data quality tools. Cichy and Rass [3] regard data quality in a particular context of an organization and compare data quality frameworks regarding definition, assessment, and improvement of data quality which are applicable with a wide range of business environments. Ehrlinger et al. [6] investigated the capabilities of 13 general-purpose data quality tools (identified with a systematic search) in detail.

Bertossi [1] discusses declarative approaches for data quality measurement, which are logic-based descriptions of what is expected to be contained in a database and specify what a result should look like. Compared to other approaches, there is no need to specify how the data cleaning and repairing is done. The logical descriptions are expressed in predicate logic with conditional functional or inclusion dependencies. Data quality systems working with conditional functional dependencies are presented in [9][2].

In [13], data quality rules for data warehouse systems are defined based on certain meta data and are implemented in SQL. Dallachiesa et al. [4] present NADEEF, which is a platform for data cleansing. The NADEEF API allows for specifying certain types of data quality rules, which describe data corruption and eventually how to repair it.

Fan [8] provides an overview of advances in the area of data quality techniques. A conditional dependency theory for capturing data problems is developed (especially duplication and incompleteness). Also, techniques for the automatic discovery of data quality rules are presented which may detect certain quality problems.

Data quality is one of the most important issues in data management. Ilyas et al. [10] provide an overview of the data cleaning process, presenting error detection and repair methods covering outlier detection, data transformation, error repair (missing values), and data de-duplication. Lettner et al. [11] present an approach on data quality management which can be integrated with existing ETL (extract transform load) infrastructures.

To the best of our knowledge, the above approaches do not provide a compact and easy syntax to define data quality rules for data entities and their relations as being presented in this work. Especially the possibility to define data quality rules based on data entities, which represent business objects rather than technical records in a database, allows to define rules in a more explicit and natural way.

3. Method

DaQL 2.0 is a extension of the domain-specific-language approach for tackling data quality originally published in [5]. In this paper, we describe the current state of development and introduce the planned innovations in the language. As in the previous version, DaQL 2.0 allows to perform data quality checks on data that are stored in relational databases. The main change of DaQL 2.0 is the introduction of entity models as the main concept on top of which data quality checks are performed.

An entity represents a business object, where the data can be distributed over several tables. The entity represents the actual object that should be tested whether it is fit for use (cf. [14]). For example, an entity for a product may be defined by the master data of the product together with all measurements performed for the product, which are stored in a separate database. Depending on the use case, entities may overlap each other and the same underlying data may turn out to be fit for use for some use cases but not for others.

An entity model is defined in DaQL 2.0 by specifying all tables and join paths between these tables that are relevant for the entity using the class `DaQLEntityModel`. This includes the definition of an `entity_key`, which defines

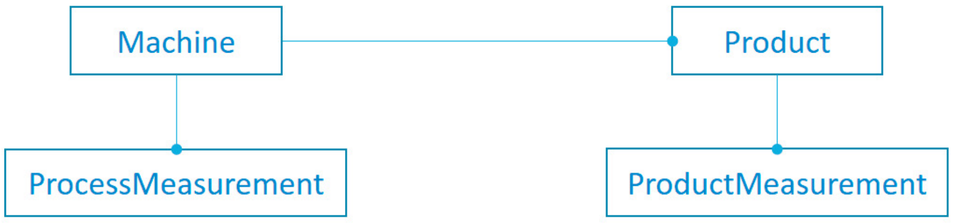

Fig. 1. Simple data model for a fictitious manufacturing process.

the granularity at which data quality checks are performed. Optionally, an `entity_orderby` expression can be specified, in case the entity key is not unique. This makes time, or the evolution of entities over time, a "first class citizen" in the DaQL 2.0 language.

Based on an entity model, data quality rules may be specified by applying a sequence of constraints that must be evaluated. A data quality constraint can be assigned to an entity model using the operation `apply_constraint(rule)`. `rule` is defined as an expression on the entity that evaluates to a Boolean value. Within a rule, the current value of every entity can be accessed using the method `current()`. If an ordering is specified for the entity model, previous values of the respective entity can be accessed using the method `prev(n)`, where n specifies the relative offset to the current entity. For each operation the mode of operation can be specified by applying the method `success()` or `failure()` on it. Success is the default mode and returns all entities that pass the data quality check, while failure returns all entities that violate the rule specified.

All operations are lazy evaluated. To trigger the evaluation, an action must be applied. DaQL 2.0 defines the following three actions: `collect()`, `count()`, or `store()`. An action `collect()` evaluates the chain of operations that define the data quality rule and returns the set of entities that pass the check, or the inverse if the failure mode is applied previously. The action `count()` also executes a data quality rule, but returns the total number of successfully evaluated entities. The action `store(name, target)` stores the evaluation results, i.e., a list of entity keys to the target specified. To ensure traceability, the name of the executed rule is also stored.

DaQL 2.0 performs all data quality checks directly on the object model generated by the object-relational (OR) mapper. This allows to formulate more expressive rules (e.g., operations on lists) compared to expressions formulated in native SQL. As a trade-off, all evaluations will be done in memory, which restricts this approach to data sets that fit into memory. This restriction would not exist for native SQL approaches.

4. Evaluation

In this section, we show how DaQL 2.0 can be used to measure data quality based on a fictitious example from a manufacturing process. The data model used is depicted in Figure 1. Products are manufactured on a machine. The table `ProcessMeasurement` contains continuously performed process measurements in regular intervals. The table `ProductMeasurement` contains measurements that are specific for individual products. Based on this example we present two data quality rules that represent cases for the data quality dimensions validity and correctness.

DaQL 2.0 is implemented in Python. The OR mapper SQLAlchemy[1] is used to access the relational database. The definition of the tables required for the entity model is provided in Listing 1. It should be noted, that the tables have to be defined only once. Based on these definitions any number of rules can be specified afterwards.

Listing 1. Definition of the tables required for the entity model.

```
Model1 = declarative_base()

class Machine(Model1):
    __tablename__ = 'Machine'
```

[1] https://www.sqlalchemy.org

```
    ID = Column(Integer, primary_key=True)
    Status = Column(String)
    StatusChangedAt = Column(DateTime)
    measurement = relationship("ProcessMeasurement", back_populates="machine")
    product = relationship("Product", back_populates="machine")

class ProcessMeasurement(Model1):
    __tablename__ = 'ProcessMeasurement'
    MachineID = Column(Integer, ForeignKey('Machine.ID'), primary_key=True)
    MeasureDate = Column(DateTime, primary_key=True)
    Metric = Column(String)
    Value = Column(Float)
    machine = relationship("Machine", back_populates="measurement")

class Product(Model1):
    __tablename__ = 'Product'
    ID = Column(Integer, primary_key=True)
    MachineID = Column(Integer, ForeignKey('Machine.ID'))
    machine = relationship("Machine", back_populates="product")
    measurement = relationship("ProductMeasurement", back_populates="product")

class ProductMeasurement(Model1):
    __tablename__ = 'ProductMeasurement'
    ProductID = Column(Integer, ForeignKey('Product.ID'), primary_key=True)
    Metric = Column(String, primary_key=True)
    Value = Column(Float)
    product = relationship("Product", back_populates="measurement")
```

4.1. Validity

This section shows how an exemplary rule to test the data quality dimension validity can be used. Specifically, it is verified, whether the measurements on a machine vary over time, i.e., a flat line is interpreted as a broken sensor. The rule is defined in Listing 2. The entity key is defined on the granularity metric (sensor). This means that every metric, including the complete available time series of measured values, represents an entity that is evaluated as a whole. The rule will be evaluated every time a new measurement is available. This example shows the usage of the methods `current()` and `previous()`.

Listing 2. Definition of rule for valid metric measurements.

```
em_machine = daql.DaQLEntityModel(
    engine=engine,
    entity_model=Model1,
    entity_key=ProcessMeasurement.Metric,
    entity_orderby=lambda x: x.MeasureDate)

test_sensor_ok = em_machine.apply_constraint(
    lambda x: x.current().Value != x.prev(1).Value)

# get list of sensors that violate the rule
test_sensor_ok.failure().collect()

# get list of products that do not violate the rule
test_sensor_ok.success().collect()
```

4.2. Correctness

The second example demonstrates a rule for the data quality dimension correctness. It checks whether the finally measured product weight stored in table ProductMeasurement equals the sum of all weights of parts added during manufacturing stored in table ProcessMeasurement. The entity key is defined at the granularity of products. The rule itself is defined in Listing 3. This example shows rules on entities, which extend over several tables.

Listing 3. Definition of rule for inconsistent weights.

```
em_machine = daql.DaQLEntityModel(
    engine=engine,
    entity_model=Model1,
    entity_key=Product.ID)

def check_weight_correct(ek):
    sum_w = 0
    for mm in ek.current().machine.measurement:
        if mm.Metric=='Weight':
            sum_w += mm.Value
    for pm in ek.current().measurement:
        if pm.Metric=='Weight':
            if pm.Value == sum_w: return True
            else: return False
    return False

test_weight = em_machine.apply_constraint(check_weight_correct)

# get list of products that violate the rule
test_weight.failure().collect()

# get list of products that do not violate the rule
test_weight.success().collect()
```

5. Deployment

DaQL 2.0 can be deployed in batch or streaming environments. In batch deployments, DaQL 2.0 is executed in regular intervals. Data is loaded directly from the relational database. Filter can be specified to restrict the amount of data loaded into DaQL (e.g., load data only from the last n days).

For streaming deployments, Apache NiFi[2] is currently supported. A custom processor evaluates data quality rules every time when new data arrives. The received data must be in JSON format and will be temporary loaded into a relational database for the evaluation of the rules. The evaluation result is returned in JSON to subsequent processors.

6. Conclusion and Future Work

The development of DaQL 2.0 is currently work in progress. Some concepts, which specifically deal with the data quality rule life cycle management, are currently under development and will be available in the final release of DaQL 2.0. Another future work is to support operations across entity models, i.e., a matching operation that can be used to detect duplicates or out of sync data stores.

[2] https://nifi.apache.org

Acknowledgements

The research reported in this paper has been funded by the Federal Ministry for Climate Action, Environment, Energy, Mobility, Innovation and Technology (BMK), the Federal Ministry for Digital and Economic Affairs (BMDW), and the Province of Upper Austria in the frame of the COMET - Competence Centers for Excellent Technologies Programme managed by Austrian Research Promotion Agency FFG.

References

[1] Bertossi, L., Bravo, L., 2013. Generic and Declarative Approaches to Data Quality Management. Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 181–211. URL: https://doi.org/10.1007/978-3-642-36257-6_9, doi:10.1007/978-3-642-36257-6_9.

[2] Chiang, F., Miller, R.J., 2008. Discovering data quality rules. Proceedings of the VLDB Endowment 1, 1166–1177.

[3] Cichy, C., Rass, S., 2019. An overview of data quality frameworks. IEEE Access 7, 24634–24648.

[4] Dallachiesa, M., Ebaid, A., Eldawy, A., Elmagarmid, A., Ilyas, I.F., Ouzzani, M., Tang, N., 2013. Nadeef: A commodity data cleaning system, in: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, Association for Computing Machinery, New York, NY, USA. p. 541–552. URL: https://doi.org/10.1145/2463676.2465327, doi:10.1145/2463676.2465327.

[5] Ehrlinger, L., Haunschmid, V., Palazzini, D., Lettner, C., 2019a. A DaQL to Monitor Data Quality in Machine Learning Applications. pp. 227–237. doi:10.1007/978-3-030-27615-7_17.

[6] Ehrlinger, L., Rusz, E., Wöß, W., 2019b. A survey of data quality measurement and monitoring tools. arXiv preprint arXiv:1907.08138 .

[7] Ehrlinger, L., Wöß, W., 2017. Automated data quality monitoring, in: Talburt, J.R. (Ed.), Proceedings of the 22nd MIT International Conference on Information Quality (ICIQ 2017), Little Rock, AR, USA. pp. 15.1–15.9.

[8] Fan, W., 2012. Data quality: Theory and practice. doi:10.1007/978-3-642-32281-5_1.

[9] Fan, W., Geerts, F., Jia, X., 2008. Semandaq: A data quality system based on conditional functional dependencies. Proc. VLDB Endow. 1, 1460–1463. URL: https://doi.org/10.14778/1454159.1454200, doi:10.14778/1454159.1454200.

[10] Ilyas, I., Chu, X., 2019. Data quality rule definition and discovery. doi:10.1145/3310205.3310211.

[11] Lettner, C., Stumptner, R., Bokesch, K.H., 2014. An approach on etl attached data quality management, in: Bellatreche, L., Mohania, M.K. (Eds.), Data Warehousing and Knowledge Discovery, Springer International Publishing, Cham. pp. 1–8.

[12] Redman, T.C., 1998. The impact of poor data quality on the typical enterprise. Communications of the ACM 41, 79–82.

[13] Rodic, J., Baranovic, M., 2009. Generating data quality rules and integration into etl process, in: Proceedings of the ACM Twelfth International Workshop on Data Warehousing and OLAP, Association for Computing Machinery, New York, NY, USA. p. 65–72. URL: https://doi.org/10.1145/1651291.1651303, doi:10.1145/1651291.1651303.

[14] Wang, R.Y., Strong, D.M., 1996. Beyond accuracy: What data quality means to data consumers. Journal of Management Information Systems 12, 5–33. arXiv:07421222.