Data Quality Checking for Machine Learning with MeSQuaL

[Demonstration paper]

Ugo Comignani
Aix Marseille Univ, Université de Toulon, CNRS, LIS, DIAMS
Marseille, France
ugo.comignani@lis-lab.fr

Noël Novelli
Aix Marseille Univ, Université de Toulon, CNRS, LIS, DIAMS
Marseille, France
noel.novelli@lis-lab.fr

Laure Berti-Équille
IRD, UMR ESPACE DEV, Montpellier
Univ de Toulon, AMU, CNRS, LIS, DIAMS, Marseille, France
laure.berti@ird.fr

ABSTRACT

This demo proposes MeSQuaL, a system for profiling and checking data quality before further tasks, such as data analytics and machine learning. MeSQuaL extends SQL for querying relational data with constraints on data quality and facilitates the verification of statistical tests. The system includes: (1) a query interpreter for SQuaL, the SQL-extended language we propose for declaring and querying data with data quality checks and statistical tests; (2) an extensible library of user-defined functions for profiling the data and computing various data quality indicators; and (3) a user interface for declaring data quality constraints, profiling data, monitoring data quality with SQuaL queries, and visualizing the results via data quality dashboards. We showcase our system in action with various scenarios on real-world data sets and show its usability for monitoring data quality over time and checking the quality of data on-demand.

1 INTRODUCTION

Assessing data quality is challenging and requires the detection and elimination of a variety of data quality problems, such as errors, duplicate, inconsistent, obsolete, and incomplete information [3, 10]. A wide range of methods for statistical analysis, constraint mining, consistency checking, and duplicate elimination has to be used [6] and their specifications can be complex for various reasons:

• *Data quality checking is a highly domain- and task-specific problem.* Data quality is multidimensional. A plethora of measurable dimensions can be used to characterize the quality of data with various indicators (e.g., value accuracy, consistency, completeness, freshness, or absence of duplicate records). Multiple techniques can be implemented to evaluate each dimension whose specification ultimately depends on the requirements of the user, the task at hand, and the application domain. Moreover, depending on the machine learning (ML) task, statistical assumptions must be verified before applying a given ML model, and the test results may ultimately influence the data preparation with a selection of specific data transformations accordingly.

• *Data quality checking is inherently a human-in-the-loop (HIL) process.* The user needs a tool offering a flexible and declarative way to define, evaluate, and check various data quality indicators and query the data with some data quality requirements in mind that can be made explicit.

• *Data quality checking is a continuous process.* Data quality may vary over time due to the temporal and dynamic nature of the data and the evolving real world, but also as a consequence of various data cleaning and repairing actions. This bears the need for monitoring the quality of different versions of the database and the quality of query results.

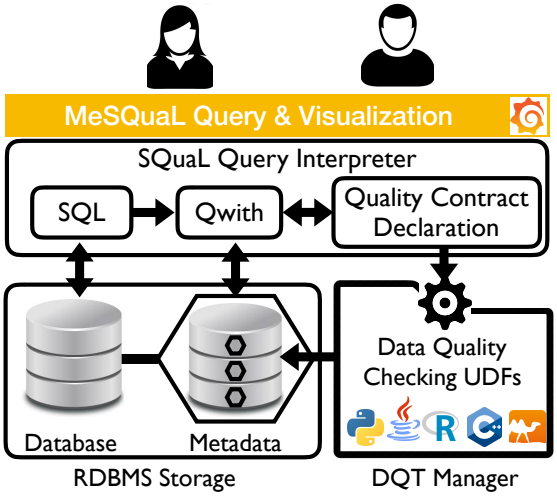

Figure 1: MeSQuaL Architecture.

Our Approach. To deal with these challenges, we believe that it is essential to build tools that enable the data scientists to specify and verify data quality requirements in a declarative way. In particular, tools for analyzing data quality, testing statistical assumptions, and monitoring data quality continuously to provide insights about the data glitches and help in selecting appropriate data preparation and cleaning strategies. In this demo, we present MeSQuaL that we built for this purpose.

2 MESQUAL OVERVIEW

Architecture. As in Fig. 1, MeSQuaL consists of two main components: (1) the SQuaL query interpreter enabling the declaration of contracts for data quality checking and SQL queries extended with QWITH statement; (2) the Data Quality and Tests (DQT) `Manager` that operates over three types of RDBMS (Oracle, MySQL, and PostgreSQL) storing the data and related metadata. Our framework provides: (1) several built-in functions for data quality checking and statistical tests; (2) the possibility to call functions in `Python`, `Java`, `C++`, `R`, and `OCaml`; or (3) the possibility to define custom command-line calls to a UDFs (User-Defined Functions). Once a quality contract is declared in SQuaL with a list of dimensions, associated UDFs, and constraints, the `DQT Manager` computes the corresponding data quality indicators and stores them as metadata. SQuaL query result and visualization are displayed via a Grafana graphical interface[1].

[1] https://grafana.com/

```
<squal-script> ::= (<squal-query> | <contract-type> | <contract>)*
<squal-query> ::= '{' <sql-query> '}'  QWITH <qwith-formula> ';'
<sql-query> ::= <select-clause><from-clause>[<where-clause>] [<group-by-clause>] [<having-clause>] [<order-by>] [<sql-sets-operator><sql-query>]* ';'
<select-clause> ::= <attribute-name> [ = <sql-subquery>]* [, <attribute-name> [ = <sql-subquery>]* ]*
<from-clause> ::= <from-element> [, <from-element> | <join-operator> <from-element>  ON <join-equality>]*
<from-element> ::= (<sql-subquery> AS <name>|<relation-name> [AS <name>])
<where-clause> ::= WHERE [NOT] EXISTS <sql-subquery> | WHERE <expression> [NOT] IN <sql-subquery>
                 | WHERE <expression> <comparison-operator> [ANY|ALL] <sql-subquery>
<having-clause> ::= <having-element> [AND <having-element>]*
<having-element> ::=  <having-expression> <comparison-operator> (<compared-value>|<sql-subquery>)
<sql-subquery> ::= (<sql-query>|<squal-query>)
<qwith-formula> ::= <qwith-element> [ AND <qwith-element>]*
<qwith-element> ::= (<contract-name> | <constraint>)
<contracttype> ::= (CREATE | REPLACE) CONTRACTTYPE <contracttype-name> <dimension> [',' <dimension>]* ';'
                 | DELETE CONTRACTTYPE <contracttype-name>  ';'
<dimension> ::= <dimension-name> BY FUNCTION <binary-path> LANGUAGE <language>
<contract> ::= (CREATE | REPLACE) CONTRACT <contract-name> '('<constraint> ('AND' <constraint>)* ');'
                 | DELETE CONTRACT <contract-name> ';'
<constraint> ::=  <contract-name> | [<contract-name> '.'] <dimension-name> <comparison-operator> <reference-value>
<comparison-operator> ::= '<>' | '=' | '!=' | '>' | '<' | '<=' | '>='
```

Figure 2: SQuaL **Syntax in EBNF**

```
CREATE CONTRACTTYPE StatTests (
   autocorrelation BY FUNCTION 'durbinWatsonTest.py' LANGUAGE PYTHON,
   multicollinearity BY FUNCTION 'varInflationFactor.py' LANGUAGE PYTHON,
   heteroscedasticity BY FUNCTION 'BreuschPaganTest.py' LANGUAGE PYTHON,
   KMerrorNormality BY FUNCTION 'KolmogorovSmirnov.py' LANGUAGE PYTHON,
   SWerrorNormality BY FUNCTION 'ShapiroWilkTest.py' LANGUAGE PYTHON);
```

```
CREATE CONTRACTTYPE CheckQDB (
   completeness BY FUNCTION 'completeness.py' LANGUAGE PYTHON,
   uniqueness BY FUNCTION 'uniqueness.py' LANGUAGE PYTHON,
   consistency BY FUNCTION 'consistency.py' LANGUAGE PYTHON,
   outlyingness BY FUNCTION 'outlyingness.py' LANGUAGE PYTHON);
```

Figure 3: CONTRACTTYPE **Examples**

```
CREATE CONTRACT RegressionAssumptions (
   StatTests.autocorrelation > 0
   AND StatTests.autocorrelation < 4
   AND StatTests.multicollinearity <= 4
   AND StatTests.heteroscedasticity < 0.05
   AND StatTests.SWerrorNormality < 0.05);
```

```
CREATE CONTRACT CheckBeforeAnalysis (
   RegressionAssumptions
   AND CheckQDB.consistency > 0.9
   AND CheckQDB.outlyingness < 0.2);
```

Figure 4: CONTRACT **Examples**

```
{   SELECT timestamp, node_id,value_raw,valuehrf
    FROM ChicagoDataset
    WHERE ChicagoDataset.sensor = 'o3'
}
QWITH CheckBeforeAnalysis
    AND CheckQDB.completeness> 0.95;
```

Figure 5: SQuaL **query example**

SQuaL Syntax. Each step of a data quality checking scenario can be expressed in SQuaL, the SQL-extended language we implemented on top of each RDBMS. The grammar of SQuaL is provided in Fig. 2. A user can easily express data quality concerns and requirements either re-using the library of UDFs we provide with MeSQuaL or, depending on his/her programming skills, adding new functions and codes in `Python`, `Java`, `C++`, `R`, or `OCaml` to check the quality of data and test other hypotheses.

• **Contract type.** Data quality measures and indicators can be expressed via the declaration of *quality contract types*. In the ⟨`contracttype`⟩ statement of the grammar in Fig. 2, a contract type statement creates (or replaces) a contract type as a list of quality dimensions of interest. Each dimension indicator is computed by a UDF. Once the *contract type* is created and UDFs are loaded, it can be instantiated as a contract with constraints on the pre-declared dimensions and later on, invoked in SQuaL queries. A dimension is defined by a ⟨*dimension-name*⟩, the path of its UDF, and the language in which the UDF is implemented. Fig. 3 presents several examples of *contract type* declarations.

• **Contract.** A quality contract, described by the ⟨*contract*⟩ statement in Fig. 2, derives from one (or more) pre-existing contract type(s) and is a set of one-sided range constraints on the dimensions declared in the contract type(s). Constraints are simple comparison expressions involving the pre-declared dimension, and a reference value. Fig. 4 presents several examples of contract declaration based on the contract types declared in the examples of Fig. 3. The `DQT Manager` executes the contracts on-demand when they are invoked in the QWITH statement of a SQuaL query.

• **Qwith Query.** In a SQuaL query, described in the grammar in Fig. 2 by the ⟨*squal-query*⟩ statement, the QWITH operator can be used to extend and constraint a regular SQL query result as illustrated in Fig. 5. It can be applied to the whole database or to a query as it adds constraints to the semantics of the SQL query and returns the query result that satisfies the quality requirements defined in the contract types and contract instances with the UDFs executed by the `DQT Manager`. Additionally, QWITH can be used inside nested SQL queries.

Note that before declaring and creating data quality contract types and contract instances using SQuaL, an important and challenging task in dealing with real-world (possibly dirty) data is data exploration to better understand the reasons for poor data quality and how it can affect the processes that consume the data. Since data exploration is out of the scope of this demo and not directly enabled by our system, we assume that data exploration should be achieved before and externally for the adequate specifications of the contract types, instances, and constraints.

Goals. In this demo, we showcase the principal features of MeSQuaL:

• *Seamless integration of user-defined functions and constraints in the query language* to compute and check various dimensions of data quality. A common way to use our system is to declare the data quality dimensions of interest, bind and invoke the functions that compute relevant quality indicators (as shown in Fig. 3 with `CheckQDB` contract type for example), and query the data with constraints on these indicators using QWITH statement of our query language (as illustrated in Fig. 5);

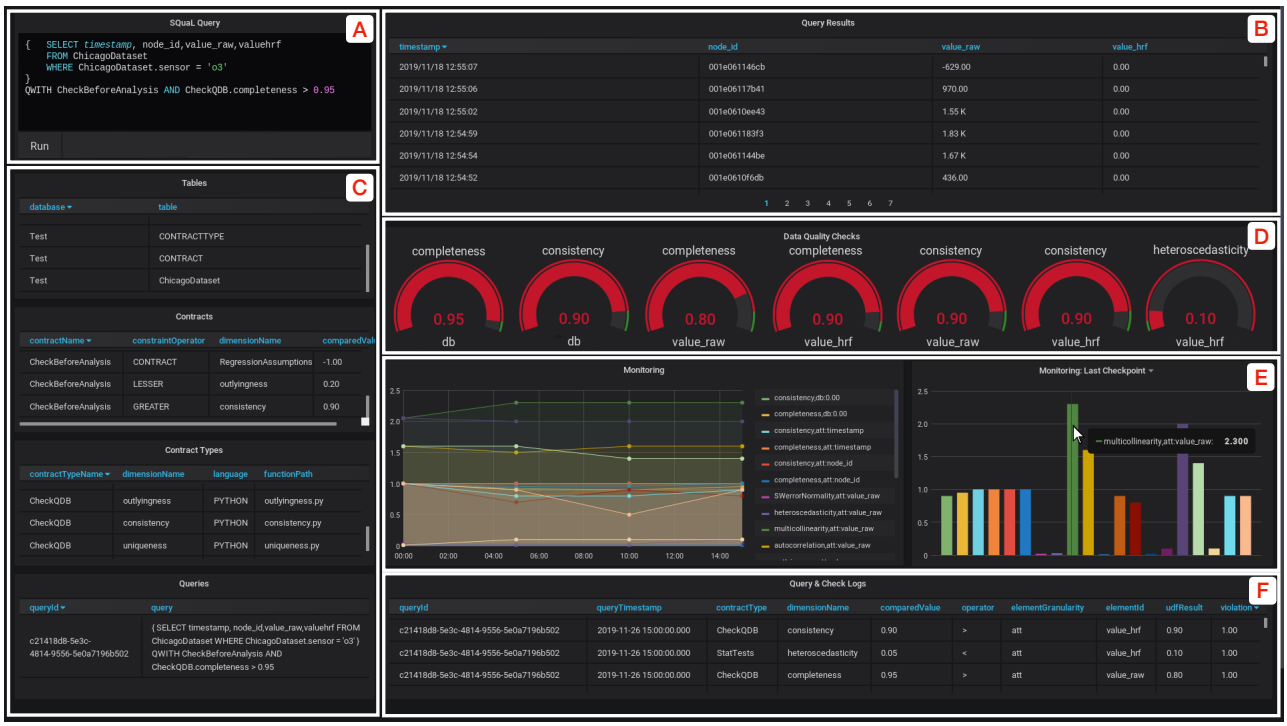

Figure 6: Screenshot of MeSQuaL's User Interface with Dynamic Dashboards

• *Continuous data-quality checking and monitoring.* MeSQuaL results can help the user in defining or comparing various methods for checking and profiling the quality of data over time, selecting the most appropriate ones, or refining the data quality checking process, and select the most appropriate contracts to monitor continuously;

• *Statistical testing.* Interactive hypothesis testing allows the user to check various statistical assumptions on the data distributions and make sure that the data conform to the requirements of a given ML model (as in Fig. 3 with `StatTests` contract type);

• *Efficient profiling of data quality indicators and visualization of static and dynamic profiles* showing the evolution of data quality over time with the Monitoring Panel E in Fig. 6.

To increase the automation of data quality checking, we claim that there is a need for augmenting database management systems with a flexible and declarative way to declare, check, and monitor the quality of data, independently from the data model, format, or application, and this actually motivated the design of MeSQuaL.

3 DEMONSTRATION SCENARIOS

We will demonstrate MeSQuaL using two domain-specific data sets: the clinical database `MIMIC-III`[2] [7] and the `ChicagoDataset` from the `Array of Things`[3] real-time urban data (AoT) [5]. The users can examine and query the data sets and explore the data quality checking functions available in MeSQuaL to gain a sense of its usability. We will guide the users through the following scenarios.

[2] https://physionet.org/content/mimiciii/1.4/
[3] https://aot-file-browser.plenar.io/data-sets

1) Declaring data quality indicators and constraints. This scenario is dedicated to showcase the use of the SQuaL language for creating and using relevant contract types and contract instances to specify data quality checks and submit SQL queries extended with QWITH statement. In Panel C of Fig. 6, the users can explore available contract types and contract instances predefined for the data set, like `CheckQDB` contract type and `CheckBeforeAnalysis` contract proposed in Fig. 3 and Fig. 4 that will check the completeness, uniqueness, consistency, or outlyingness of the data. The users will be able to declare new data quality checks and rules, and query the data sets with SQuaL in Panel A. For the query of Fig. 5, Panel B presents the results on `ChicagoDataset` and red gauges of Panel D show that neither the constraints on data consistency and completeness are satisfied by the queried data of ozone sensors ('o3'), nor the constraints defined in `RegressionAssumptions` contract. As presented in Table 1, other SQuaL queries including nested SQuaL queries (e.g., Q_8) will be tested to show the usability of our system.

2) Evaluating the applicability of learning models with declarative statistical hypothesis testing. In this scenario, the user will see in more detail how MeSQuaL can be used to declare various statistical tests, and visualize which data lead to violations of some statistical assumptions or other requirements of various ML models. For example, before the application of a linear regression over a data set, at least four critical assumptions need to be verified: normality, linearity, homoscedasticity, and absence of multicollinearity. Using `StatTests` contract type instantiated by `RegressionAssumptions` contract (defined in Fig. 3 and Fig. 4) available in the contract explorer (Panel C), the user will easily check if the statistical properties are met by the query results. The attendees will notice that new contracts can be added in a flexible and modular way, and they will explore our library of tests, inspect the logs of previous queries and

Table 1: SQuaL Query examples of the demo

	Query#	SQuaL Query	SQuaL Query time (ms)	SQL query time (ms)	UDF time (ms)	Total time (ms)
	Q_1	CREATE CONTRACTTYPE CheckQDB2 (completeness2 FLOAT ON DATABASE BY FUNCTION 'completeness.py' LANGUAGE PYTHON, uniqueness2 FLOAT ON DATABASE BY FUNCTION 'uniqueness.py' LANGUAGE PYTHON, consistency2 FLOAT ON DATABASE BY FUNCTION 'consistency.py' LANGUAGE PYTHON, outlyingness2 FLOAT ON DATABASE BY FUNCTION 'outlyingness.py' LANGUAGE PYTHON);	24.6	-	2482.4	2507
	Q_2	CREATE CONTRACT RegressionAssumptions (StatTests.autocorrelation > 0 AND StatTests.autocorrelation < 4 AND StatTests.multicollinearity <= 4 AND StatTests.heteroscedasticity < 0.05 AND StatTests.SWerrorNormality < 0.05);	28.3	-	2550.8	2579.1
AoT	Q_3	{ SELECT * FROM ChicagoDataset } QWITH CheckQDB.completeness> 0.95;	32.9	587.2	566.2	1186.3
AoT	Q_4	{ SELECT * FROM ChicagoDataset } QWITH CheckBeforeAnalysis AND RegressionAssumptions;	274.6	499.6	500.4	1274.6
AoT	Q_5	{ SELECT timestamp, node_id,value_raw,valuehrf FROM ChicagoDataset WHERE ChicagoDataset.sensor = 'o3' } QWITH CheckBeforeAnalysis AND CheckQDB.completeness> 0.95;	214.2	21.5	784.7	1020.4
MIMIC-III	Q_6	{ SELECT * FROM Admissions } QWITH CheckQDB.completeness> 0.95;	42.1	346.8	472.3	861.2
MIMIC-III	Q_7	{ SELECT * FROM Admissions WHERE Admissions.insurance = 'Private' } QWITH CheckBeforeAnalysis AND CheckQDB.completeness> 0.95;	237.0	154.1	493.8	884.9
MIMIC-III	Q_8	{ SELECT gender, dob, admittime FROM Admissions INNER JOIN (SELECT * FROM Patients WHERE dob < '2090-12-12 00:00:00' QWITH CheckQDB.completeness> 0.95) as Pat ON Admissions.subject_id=Pat.subject_id; } QWITH CheckQDB.completeness> 0.95;	318.9	16.6	1610.1	1945.6

checks (Panel F), and reload the visualization of some previous SQuaL queries (from Panel C). MeSQuaL combines the declarative and scripting approaches for querying the data and checking various assumptions simultaneously. It facilitates notably the data preparation choices to meet the requirements of some ML algorithms. Since the declared contracts are independent of the data sets, they can be reused whenever the assumptions and data quality checks need to be checked.

3) Monitoring the evolution of data quality indicators. In this last scenario, the attendees will see the possibilities offered by MeSQuaL for monitoring the evolution of data quality indicators in Panel E. Once declared, contract types and contract instances are stored as metadata and executed regularly by MeSQuaL's `DQT Manager` via its configuration to schedule recurring SQuaL queries. The user can act as a DBA and define various thresholds for the declared data quality indicators to alert when some results are suspicious. Using the `AoT ChicagoDataset`, the attendees will see how MeSQuaL facilitates the continuous monitoring of data quality indicators to detect, for instance, inconsistent data from neighboring air pollution sensors or intermittent sensor failures with the use of the library of UDFs provided by MeSQuaL. Using MeSQuaL's user interface shown in Fig. 6, the user can visualize the declared data quality indicators and spot the periodic or punctual errors in the data over time in Panel E.

4 RELATED WORK

Data quality has been extensively studied by the database community in the last decades [3, 10] with a line of data cleansing commodity systems and tools that can detect anomalies and reduce the burden on data scientists for data repairing and data preparation in the context of ML pipelines [1, 2, 8, 9, 13]. MeSQuaL is similar to some extent to two main operational data validation systems: (1) Google TensorFlow Data Validation (TFDV) system [4], used in production, is a library for exploring and validating ML data, including schema inspection and anomaly detection, such as missing features, out-of-range values, or wrong feature types, and (2) the Amazon system implementing unit-tests for data verification has been proposed in [11, 12]; it offers a declarative API that allows users to define checks on their data by composing a variety of available constraints. However, the main drawback of these systems is that they do not provide the user with (1) the possibility to interact with the data quality checking process, (2) the flexibility to declare new data quality metrics with user-defined functions, or constraints for data quality checks, and (3) the extension of the query language to check the results with respect to data quality requirements and constraints.

The key novelty of MeSQuaL is to provide the user with a framework for checking the quality of their relational data by declaring UDFs and constraints using SQuaL, an SQL-like query language extension for querying data and checking on-demand the quality of the results.

5 ACKNOWLEDGEMENTS

This work is funded by the French National Agency ANR Quali-Health 18-CE23-0002.

REFERENCES

[1] L. Berti-Équille. Learn2Clean: Optimizing the Sequence of Tasks for Web Data Preparation. In *Proc. of the The Web Conf 2019*, 2019.
[2] L. Berti-Équille. Reinforcement learning for data preparation with active reward learning. In *Internet Science - 6th International Conference, INSCI 2019, Perpignan, France, December 2-5, 2019, Proceedings*, pages 121–132, 2019.
[3] L. Berti-Équille. Quality Awareness for Data Management and Mining. Habilitation à Diriger des Recherches, Univ. Rennes 1, France, June 2007, http://pageperso.lis-lab.fr/~laure.berti/pub/Habilitation-Laure-Berti-Equille.pdf.
[4] E. Breck, M. Zinkevich, N. Polyzotis, S. Whang, and S. Roy. Data validation for machine learning. In *Proc. of SysML*, 2019.
[5] C. E. Catlett, P. H. Beckman, R. Sankaran, and K. K. Galvin. Array of things: A scientific research instrument in the public way: Platform design and early lessons learned. In *Proc. of the 2nd International Workshop on Science of Smart City Operations and Platforms Engineering*, SCOPE '17, pages 26–33, New York, NY, USA, 2017. ACM.
[6] I. F. Ilyas and X. Chu. Trends in cleaning relational data: Consistency and deduplication. *Foundations and Trends in Databases*, 5(4):281–393, 2015.
[7] A. E. Johnson, T. J. Pollard, L. Shen, H. L. Li-wei, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. A. Celi, and R. G. Mark. MIMIC-III, a freely accessible critical care database. *Scientific data*, 3:160035, 2016.
[8] S. Krishnan, J. Wang, E. Wu, M. J. Franklin, and K. Goldberg. Activeclean: Interactive data cleaning for statistical modeling. *Proc. VLDB Endow.*, 9(12):948–959, Aug. 2016.
[9] T. Rekatsinas, X. Chu, I. F. Ilyas, and C. Ré. Holoclean: Holistic data repairs with probabilistic inference. *PVLDB*, 10(11):1190–1201, 2017.
[10] S. W. Sadiq, T. Dasu, X. L. Dong, J. Freire, I. F. Ilyas, S. Link, M. J. Miller, F. Naumann, X. Zhou, and D. Srivastava. Data quality: The role of empiricism. *SIGMOD Record*, 46(4):35–43, 2017.
[11] S. Schelter, F. Biessmann, D. Lange, T. Rukat, P. Schmidt, S. Seufert, P. Brunelle, and A. Taptunov. Unit testing data with deequ. In *Proc. of the 2019 International Conference on Management of Data*, SIGMOD '19, pages 1993–1996, New York, NY, USA, 2019. ACM.
[12] S. Schelter, D. Lange, P. Schmidt, M. Celikel, F. Bießmann, and A. Grafberger. Automating large-scale data quality verification. *PVLDB*, 11(12):1781–1794, 2018.
[13] M. Yakout, L. Berti-Équille, and A. K. Elmagarmid. Don't be scared: use scalable automatic repairing with maximal likelihood and bounded changes. In *Proc. of the ACM SIGMOD*, pages 553–564, 2013.