DetCat: Detecting Categorical Outliers in Relational Datasets

Arthur Zylinski
a.zylinski@students.uu.nl
Utrecht University
Utrecht, The Netherlands

Abdulhakim A. Qahtan*
a.a.a.qahtan@uu.nl
Utrecht University
Utrecht, The Netherlands

ABSTRACT

Poor data quality significantly affects different data analytics tasks, leading to inaccurate decisions and poor predictions of the machine learning models. Outliers represent one of the most common data glitches that impact data quality. While detecting outliers in numerical data has been extensively studied, few attempts were made to solve the problem of detecting categorical outliers. In this paper, we introduce DetCat for detecting categorical outliers in relational datasets, by utilizing the syntactic structure of the values. For a given attribute, DetCat identifies a set of patterns that represents the majority of the values as dominating patterns. Data values that cannot be generated by the dominating patterns are declared as outliers. The demo will show the effectiveness of our tool in detecting categorical outliers and discovering the syntactical data patterns.

PVLDB Reference Format:
Arthur Zylinski and Abdulhakim A. Qahtan. DetCat: Detecting Categorical Outliers in Relational Datasets. PVLDB, XX(XX): XXX-XXX, 2024.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/qahtanaa/DetectiveCat.git.

1 INTRODUCTION

Poor data quality is a challenging problem that researchers and data scientists have to deal with during their daily work. Studies reported that around 50-80% of the working time of data scientists is spent on what is called "janitor work", such as finding datasets, profiling, cleaning, and wrangling [3]. These activities are mandatory for preparing the data before performing any data analytics task.

Data anomalies represent one of the common data quality issues. Even though detecting anomalies in attributes with numerical values has been studied extensively, detecting anomalies in categorical attributes is less mature due to the difficulty in defining the similarity between the values. Therefore, it is important to develop a tool that can identify anomalies in attributes with categorical values.

For relational data, values in categorical attributes usually follow specific syntactic structures. Hence, anomalies can be identified as the values that do not conform with the syntactic structure of the majority. For example, a *date* attribute follows different syntactic structures such as *dd-mm-yyyy*, *dd-mm-yy* or *dd/mm/yyyy*, etc. For such an attribute, values that have the year only *yyyy* can be reported as outlier. This property of the attributes in relational data was utilized by FAHES [5] for detecting disguised missing values (DMVs) and SURAGH [1] for reporting ill formed records. However, FAHES focuses on the DMVs and SURAGH reports the whole record without identifying the actual erroneous cell. Therefore these tools are not suitable for detecting categorical outliers in general.

In this demo paper, we describe the DetCat (Detecting Categorical outliers) tool that we implemented for detecting the anomalies in attributes with categorical values. DetCat utilizes the syntactic structure of the values in order to identify a set of dominant patterns (Reg-Ex like patterns) that generates the majority (maybe all) of the values in the attribute. Values that cannot be generated by the dominant patterns are reported as anomalies (outliers). We will use the terms anomalies and outliers interchangeably. For defining the Reg-Ex like patterns, the general class of regular expression can be used. However, this class is too large for the purpose of anomaly detection. In addition, it complicates the problems (i.e. high time complexity) of discovering the dominating patterns, e.g. checking the equivalence of two regular expressions is PSPACE-complete [7]. In our case, simple patterns are typically sufficient, as it has been shown in [5]. The benefit of defining simple patterns instead of complex regular expressions is that they are easy to specify, discover, apply, reason about, and most importantly, they are enough for detecting the anomalies [4]. The demo will show the effectiveness of DetCat in discovering the different patterns in datasets from diverse domains and detecting the outliers in the categorical attributes that cannot be detected otherwise.

2 SYNTACTIC PATTERNS

Before describing our method in discovering the dominating patterns, we need to define them formally. Moreover we need to distinguish between expressive and generic patterns.

2.1 Preliminaries

To identify a set of simple representative patterns of the data, we utilize the *generalization tree* (GT) that is shown in Figure 1. If we denote the set of all the characters that can be entered by the keyboard as $\mathcal{I}$, then we define the *generalization tree* as a tree over an alphabet $\Sigma = \mathcal{I} \cup \tilde{\mathcal{I}}$, where each leaf node is a character in $\mathcal{I}$ and each intermediate node is a generalization (class), from the set $\tilde{\mathcal{I}} = \{$Ů, Ł, Đ, Š, ⊔$\}$, of its child nodes. These classes represent a partitioning for the alphabet. The classes are: upper case letters (Ů = [A-Z]), lower case letters (Ł = [a-z]), digits (Đ= [0-9]), white spaces (⊔= [new line, tab, space]) and symbols (Š = {#, @, &, :, ; , ...}).

We also define a *pattern*, denoted by P, as a sequence of characters over the generalization tree (GT). We say that a string s *matches* (can be generated by) a pattern P, denoted by $s \mapsto P$, if s is evaluated to be *true* by P (i.e. the value satisfies the pattern definition). For

Proceedings of the VLDB Endowment, Vol. XX, No. XX ISSN 2150-8097.
doi:XX.XX/XXX.XX

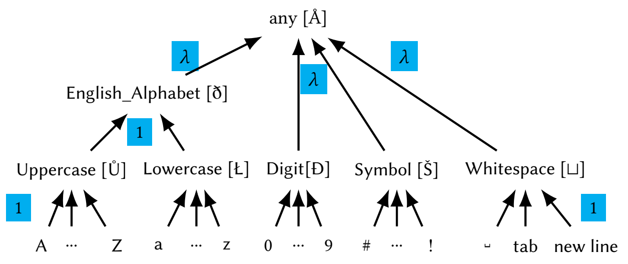


Figure 1: The generalization tree (GT). The weights on the edges from the leaves to their parents is 1, while the weight on the edges from the parents to the root node is λ. These weights will be used to define a modified version of the minimum edit distance in Section 3.3.

example, 90001 $\mapsto$ ĐĐĐĐĐ. We define the *language of a pattern*, denoted by $\mathcal{L}(P)$, as the set of all strings that matches P. Moreover, we define the *pattern coverage* for a pattern P that is extracted from the values of an attribute A, denoted by $C(P)$, as the percentage of the values $v \in A$ that match P. That is $C(P) = \frac{|\{v \in A : v \mapsto P\}|}{|A|}$.

2.2 Expressive and Generalized Patterns

Given two patterns P and P', P is said to be more *expressive* pattern than P', if for any string s, $s \mapsto P$ implies $s \mapsto P'$. That is $\mathcal{L}(P) \subseteq \mathcal{L}(P')$. Consequently, pattern P' is a *generalized* pattern of P. The expressiveness and generalization can be associated with the ratio of the characters from the leaf node of the generalization tree that exist in the pattern. Intuitively, the expressive pattern should include more characters from the leaf nodes compared to the generalized pattern. For example, consider the zip code 90001 and the two patterns P_1 = ĐĐĐĐĐ and P_2 = 900ĐĐ. We can see that 90001 $\mapsto P_1$, 90001 $\mapsto P_2$, and $\mathcal{L}(P_2) \subseteq \mathcal{L}(P_1)$. Here, P_2 is said to be more expressive than P_1 and P_1 is a generalization of P_2.

The selection between expressive and generalized patterns has an impact on the efficiency and the accuracy of DetCat. On one hand, constructing generic patterns from the values of a given attribute will ensure a compact representation of the attribute, as less patterns will be discovered. The limited number of generalized patterns, however, might hide additional interesting information, leading to potentially many categorical outliers going undetected. On the other hand, the number of unique values that matches an expressive patterns might be small, which would require discovering numerous patterns to fully cover the values in the attribute. For DetCat, we use a scoring function that simultaneously considers the coverage and the expressiveness of the patterns.

3 THE SYSTEM ARCHITECTURE

The DetCat architecture is shown in Figure 2. The tool consists of multiple components. In the following subsections, we discuss these components in details.

3.1 Numerical Attributes Pruning

Initially, summary statistics about the data are collected and the types of values in each attribute are identified. We identify four types of the data {*integer, float, text, code*}. Inspired by [4], we remove

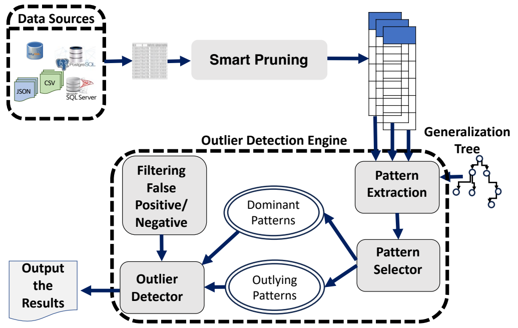


Figure 2: The DetCat Architecture.

the pure numerical attributes that represent *quantitative values*. By quantitative, we refer to the values that are commonly used in statistics, which typically has meaning as a measurement (e.g., a person's height) or a count (e.g., a person's stock shares). These attributes are identified to be of type 'integer' or 'float'. In contrast, we study the *qualitative values* even if they have numerical representation. Qualitative values are non-statistical and they cannot be aggregated. For example, it is meaningless to compute the average of two zip codes. The qualitative attributes are identified as 'code' or 'text' by DetCat. The 'code' attributes contain values with the same length for the majority of the values, whereas the values in the 'text' attributes would have arbitrary length.

3.2 Pattern Generation

After profiling the data and discarding the quantitative attributes, we generate a set of representative Reg-Ex like expressions (patterns), that can generate all the values in the attribute, for each of the remaining qualitative attributes. The patterns are generated by replacing zero or more characters in each value with the characters of their parent nodes in the GT (Figure 1)). Examples of the patterns that can be generated from 'leaf' and 'US$' are presented in Table 1.

The number of characters from the leaf nodes of the tree in each pattern determines the expressiveness level of that pattern. In this work, we aim to have a minimal number of representative patterns with maximum possible level of expressiveness. For this reason, we generate all the possible patterns from each value. Since the number of possible patterns that can be generated from a value of length n is 2^n, we use multi-level index for fast access to the generated patterns. First, we index the patterns based on their length. After that, we create a hash table for the different patterns with the same length. Moreover, we store the number of values that matches each pattern to identify the dominant patterns afterwards.

After generating all the possible patterns from an attribute A, we maintain a minimal set of representative patterns $\mathfrak{P} = \{P_1, \ldots, P_k\}$. The selection is done using a scoring function that considers the trade-off between the expressiveness and compactness of the set of patterns. The score of a given pattern P is define as: $s(P) = \frac{C(P)}{|\mathfrak{C}(P)|+1}$, where $C(P)$ is the coverage of pattern P and $\mathfrak{C}(P) = \{\Upsilon \in P \wedge \Upsilon \in \tilde{\Sigma}\}$. For example, if we have an attribute "year" that contains 120 values, where 96 values start with '19' and 108 values start with '1', then the score of P_1 = 19ĐĐ and p_2 = 1ĐĐĐ are $s(P_1) = \frac{32/120}{3} = 0.089$

Table 1: Examples of the generated patterns.

Literal String	Possible Patterns
leaf	leaf, leaŁ, leŁf, leŁŁ, lŁaf, lŁaŁ, lŁŁf, lŁŁŁ, Łeaf, ŁeaŁ, ŁeŁf, ŁeŁŁ, ŁŁaf, ŁŁaŁ, ŁŁŁf, ŁŁŁŁ
US$	US$, USŠ, UŮ$, UŮŠ, ŮS$, ŮSŠ, ŮŮ$, ŮŮŠ

and $s(P_2) = \frac{27/120}{4} = 0.056$, respectively. In this case, the pattern P_1 will be chosen over P_2.

The set of patterns $\mathfrak{P}$ is *comprehensive* in the sense that each value in the attribute A matches one of the patterns in $\mathfrak{P}$. That is: $\forall a \in A, \exists P_i \in \mathfrak{P}\ s.t.\ a \mapsto P_i$. Moreover, the set $\mathfrak{P}$ contains *disjoint* patterns where each value $a \in A$ can match only one pattern in $\mathfrak{P}$. That is, $\forall a \in A, a \mapsto P_i \wedge a \mapsto P_j \implies i = j$.

When the number of representative patterns is getting large, we merge patterns that have high similarity to reduce the number of patterns. However, since the patterns may include characters from the intermediate nodes of the GT, traditional string similarity metrics will not work for comparing two patterns. Because of that, we propose a modified version of the Levenshtein Minimum Edit Distance (LMED) [2] in Section 3.3. Using the modified minimum edit distance, we compute the distance between the different patterns and merge the patterns that are very similar to each other. Initially, we merge the patterns that have the same length since they are in the same hash-table. We merge patterns with different lengths only during the false positive/negative reduction step in Section 3.5.

Moreover, for attributes with long strings and high diversity in the characters that form the values in the attribute, only two patterns are generated. The first pattern contains the characters from the leaves of the GT (i.e. the actual value). For the second pattern, we replace each character from a given value with the parent of the leaf that contains the character. This heuristic reduces the expressiveness level of the generated patterns, but improves the efficiency of the tool significantly. For large tables, we sample a representative set of values and use them to generate an initial set of patterns. More patterns are added when necessary after comparing the rest of the values against the initial set of patterns.

3.3 Modified Min-Edit Distance (MMED)

For computing the distance between the different patterns, we propose a modified version of the minimum edit distance (MMED) that defines the cost for substituting a character α with character δ as the cost for traversing the generalization tree from node α to node δ. Formally speaking, if the path between nodes α and δ is $\varkappa$, then the cost for substituting α by δ is computed as $d_s(\alpha, \delta) = \sum_{e \in \varkappa} \omega_e$, where d_s is the substitution cost, e is an edge on the path $\varkappa$ and $\omega_e \in \{1, \lambda\}$ is the weight associated with the edge e. We choose $\lambda > 1$ in order to increase the distance between patterns with characters from different branches around the root of the tree.

The cost for inserting/deleting a character α between two characters δ and ρ, denoted by $d_{id}(\alpha, \{\delta, \rho\})$, is define as: $d_{id}(\alpha, \{\delta, \rho\}) = d_s(\alpha, \Gamma_{\alpha,\delta}) + d_s(\alpha, \Gamma_{\alpha,\rho})$, where $\Gamma_{\alpha,\delta}$ is the common parent of α, δ and $\Gamma_{\alpha,\rho}$ is the common parent of α, ρ. Here, we consider the 'null' character as a leaf in each branch of the GT. In this case, inserting α at the beginning or the end of a value will be computed as

Table 2: Examples of the distance between different patterns.

S1, S2	d(S1, S2)	Notes
ĐĐĐĐ, 1234	4	$d_s(Đ, 1) + d_s(Đ, 2) + d_s(Đ, 3) + d_s(Đ, 4)$
Alex, Adam	6	$d_s(l, d) + d_s(e, a) + d_s(x, m)$
Alex, 3lex	7	$d_s(A, 3)$
Alex, alex	4	$d_s(A, a)$
Alex, A3lex	6	$d_{id}(3, \{A, l\}) = d_s(3, \Gamma_{3,A}) + d_s(3, \Gamma_{3,l})$
Alex, Aex	3	$d_{id}(l, \{A, e\}) = d_s(l, A) + d_s(l, e) = 2 + 1$
ĐĐĐ, 1234	5	$d_s(Đ, 1) + d_s(Đ, 2) + d_s(Đ, 3) + d_{id}(4, \{null, Đ\})$

$d_{id}(\alpha, \{null, \delta\}) = d_s(\alpha, \Gamma_\alpha) + d_s(\alpha, \Gamma_{\alpha,\delta})$, where Γ_α is the parent node of α. Examples for computing the distance between different patterns when $\lambda = 2$ are available in Table 2.

3.4 Dominant Pattern Identification

After generating the set of representative patterns $\mathfrak{P}$ that can generate all the values in the attribute, we identify the set of *dominant patterns* $\tilde{\mathfrak{P}}$. To do so, we start by sorting the representative patterns according to their score. Then, we add the top patterns to the set $\tilde{\mathfrak{P}}$ until the ratio of values from the attribute that match one of the patterns in $\tilde{\mathfrak{P}}$ is greater than a user specified threshold ψ. The recommended value for ψ is ≥ 0.95. This recommendation is based on the study in [6] where the ratio of erroneous values is estimated to be around '5%' in many datasets. Moreover, according to the normality assumption of the data in statistics and using the 3σ rule, 99.7% of the data are considered to be normal. However, for a pattern P_k to be added to the set of dominant patterns $\tilde{\mathfrak{P}}$, its coverage $C(P_k) \geq \gamma \times \frac{\sum_{P_i \in \mathfrak{P}} C(P_i)}{|\mathfrak{P}|}$. The rest of the patterns are flagged as potential outliers and passed with the dominating patterns to a module that filters the false positives/negatives.

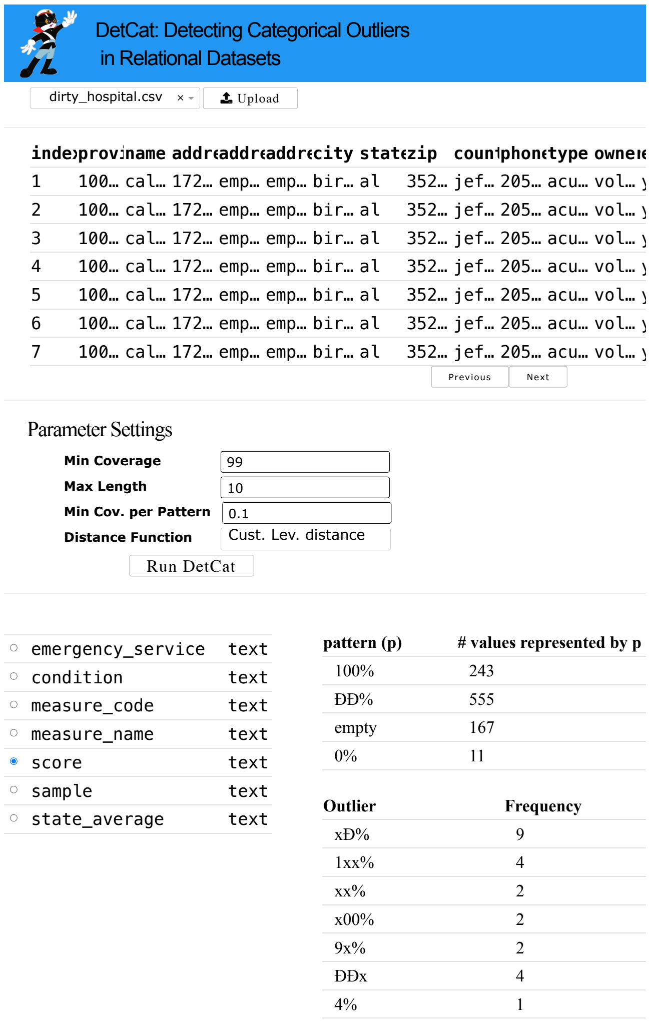
3.5 Eliminating False Predictions

After determining the set of dominating patterns, we use another module to minimize the number of values that are reported incorrectly (false positives/negatives). To do so, we use the MMED to compare the patterns that are close to the cutoff threshold with the patterns that are clear dominant/outlying patterns. Patterns with small average distance to the dominant patterns are considered as normal patterns and vice versa. In Section 3.2, we considered merging patterns with the same length by replacing characters in multiple patterns with the character in their parent node. Here, we consider the possibility to merge patterns with multiple insertions from the same node and replace them using the '+' symbol, where α+ means one or more repetitions. For example, the patterns {ŮŁ, ŮŁŁ, ŮŁŁŁŁ, ŮŁŁŁŁŁŁ} can be merged into a single pattern ŮŁ+. Consequently, if one of these patterns is in $\tilde{\mathfrak{P}}$ then ŮŁ+ $\in \tilde{\mathfrak{P}}$.

4 DEMONSTRATION PLAN

During the demo, we will use publicly available datasets that are used as benchmarks for evaluating the data cleaning tools. We will use the datasets that were used by RAHA. However, the audience is also welcome to bring their own datasets to test our tool.

[1]https://github.com/qahtanaa/DetCat.git


DetCat: Detecting Categorical Outliers in Relational Datasets

dirty_hospital.csv × Upload

inde	prov	name	addr	addr	addr	city	stat	zip	coun	phon	type	owne	e
1	100…	cal…	172…	emp…	emp…	bir…	al	352…	jef…	205…	acu…	vol…	y
2	100…	cal…	172…	emp…	emp…	bir…	al	352…	jef…	205…	acu…	vol…	y
3	100…	cal…	172…	emp…	emp…	bir…	al	352…	jef…	205…	acu…	vol…	y
4	100…	cal…	172…	emp…	emp…	bir…	al	352…	jef…	205…	acu…	vol…	y
5	100…	cal…	172…	emp…	emp…	bir…	al	352…	jef…	205…	acu…	vol…	y
6	100…	cal…	172…	emp…	emp…	bir…	al	352…	jef…	205…	acu…	vol…	y
7	100…	cal…	172…	emp…	emp…	bir…	al	352…	jef…	205…	acu…	vol…	y

Parameter Settings

Min Coverage	99
Max Length	10
Min Cov. per Pattern	0.1
Distance Function	Cust. Lev. distance

○	emergency_service	text
○	condition	text
○	measure_code	text
○	measure_name	text
●	score	text
○	sample	text
○	state_average	text

pattern (p)	# values represented by p
100%	243
ÐÐ%	555
empty	167
0%	11

Outlier	Frequency
xÐ%	9
1xx%	4
xx%	2
x00%	2
9x%	2
ÐÐx	4
4%	1

Figure 3: The interface of the tool. The Video shows more details about running the tool and setting the parameters[1].

4.1 Patterns Discovery and Outlier Detection

Figure 3 shows the interface of DetCat, where a snippet of the hospital[2] dataset is shown for the user to give an understanding about the dataset. Attributes that are identified as 'text' or 'code' are shown in the table on the bottom-left corner with the type of each attribute. The selection buttons next to the names of the attributes allow the user to display the dominant patterns and the outliers in the selected attribute. The figure shows that for the attribute 'score', there are a few dominant patterns {'100%', 'empty', 'ÐÐ%', 0%'}. The identified outliers are also displayed with their frequencies. The pattern '4%' clearly represents a normal value, but it has been identified as an outlier. However, this pattern has been flagged as a false positive pattern by the false positive/negative filtering module. Other patterns and outliers that were discovered by DetCat from the dirty version of the hospital dataset are shown in Table 3. It is interesting that DetCat reported the zip code {99559} as an outlier since the discovered dominating patterns are {'3ÐÐÐÐ', '99508', 'xÐÐÐÐ'}. While this code is a correct zip code, it belongs to a different state (Alaska) than the state (Alabama) of the majority of the values in the dataset. Moreover, the code 'xÐÐÐÐ' represents a false negative and fooled our tool since it represents a reasonable number of values.

[2]https://github.com/BigDaMa/raha/tree/master/datasets/hospital

Table 3: Examples of the discovered erroneous patterns.

attribute	Dominant Patterns	Erroneous Patterns
provider_number	100ÐÐ	x00ÐÐ, 100xÐ, 100Ðx
city	birmingham, gadsden, dothan	faxette, annxston, hamilxon
zip	3ÐÐÐÐ, 99508	3x640, x590x, 99559
sample	Ð patients, ÐÐ patients, ÐÐÐ patients, empty	x patients, 15 patiexts, 126 patxents

4.2 Effects of the Configuration Parameters

The conference attendees can also check the effects of changing the configuration parameters on the performance of our tool. Detcat has four main parameters: i) minimum coverage of the dominant patterns (ξ): we add more patterns to the set of dominant patterns until the ratio of the generated values using the dominant patterns to the total number of values is greater than ξ; the rest of the patterns are flagged as potential erroneous patterns. ii) The max allowed length for values (L): no exhaustive search for patterns in values with length greater than L; instead, we use only two patterns (the pattern of the original characters and the fully generalized pattern). iii) The minimum coverage per dominant pattern (γ): for a pattern p_i to be flagged as dominant pattern, the ratio of the generated values by p_i to the total number of values in the attribute must be greater than γ; Parameters ξ and γ are considered based on the fact that outliers are rare events and should be a minority of any given dataset. iv) The similarity measure: we use the customized Levenshtein distance to measure the distance between patterns. However, attendees can use the original Levenshtein distance to filter the false positives/negatives. This can show the power of our new distance measure in filtering the false predictions. The demo will provide a clear understanding of outlier detection in attributes with categorical values.

REFERENCES

[1] Mazhar Hameed, Gerardo Vitagliano, Lan Jiang, and Felix Naumann. 2022. SURAGH: Syntactic Pattern Matching to Identify Ill-Formed Records. In *(EDBT'22)*. 143–154.

[2] Vladimir I Levenshtein et al. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, Vol. 10. Soviet Union, 707–710.

[3] Steve Lohr. 2014. For Big-Data Scientists, 'Janitor Work' Is Key Hurdle to Insights. *New York Times* (Aug 2014).

[4] Abdulhakim Qahtan, Nan Tang, Mourad Ouzzani, Yang Cao, and Michael Stonebraker. 2020. Pattern Functional Dependencies for Data Cleaning. *Proc. VLDB Endow.* 13, 5 (2020), 684–697.

[5] Abdulhakim A. Qahtan, Ahmed Elmagarmid, Raul Castro Fernandez, Mourad Ouzzani, and Nan Tang. 2018. FAHES: A Robust Disguised Missing Values Detector. In *SIGKDD'18*. 2100–2109.

[6] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. 2017. Holoclean: Holistic data repairs with probabilistic inference. *Proc. VLDB Endow.* 10, 11 (2017), 1190–1201.

[7] Larry J. Stockmeyer and Albert R. Meyer. 1973. Word Problems Requiring Exponential Time: Preliminary Report. In *STOC*. 1–9.