Lakehouse – Data Integration: Data Profiling – Binary Datatype Analysis

by
Kevin Gieswein

Supervisors
Prof. Dr. Thorsten Papenbrock
Alexander Vielhauer

Philipps Universität Marburg

March 24, 2023

Abstract

This work mainly deals with the identification of binary semantic data types in large collections of relational databases. We aim to calculate a score that indicates how well a candidate can be identified as a binary semantic data type. To achieve this, we use various heuristics, which we then weigh differently to obtain the best possible result.

The identification of semantic data types belongs to the field of data profiling. Through this identification, we can classify our data into concrete semantic classes. Not only does this classification help us to better understand the meaning of our data, but it also enables us to convert the data into uniform formats. The conversion into uniform formats and classes is an essential factor in later data cleaning. Besides that semantic data types are also a huge part of proper schema matching.

Contents

1 Introduction

This work mainly deals with the automatic detection of binary semantic data types in big relational data sets. This means that we consider semantic data types that can be divided into two categories as precisely as possible. These two categories should be as contrasting as possible. (see definition 1)

In this work, we will first discuss how we identify candidates. Then we will define heuristics by which we will evaluate our candidates. Next, we will define how the score is calculated, which should indicate the rank a candidate should have. The rank should ultimately be a value between 0 and 1, indicating how likely it is that a candidate is a binary data type.

At the end, we will test our score on a 19.1GB relational data set with 1,018,141 tables from gittables (see bibliography [2]) and see how well it performs.

1.0.1 Related Work

The identification of semantic data types has been researched to some extent, with several studies exploring various methods for achieving this. Most studies in this area nowadays adopt an approach based on neural networks, including works such as "Sherlock: A Deep Learning Approach to Semantic Data Type Detection" (see bibliography [1]) and "DCoM: A Deep Column Mapper for Semantic Data Type Detection" (see bibliography [3]). These scientific papers focus on using large data sets as training data for a neural network, with the goal of training artificial intelligence to recognize and identify semantic data types.

However, there are also approaches to identifying semantic data types without the use of neural networks. An example of this is presented in the paper "Synthesizing Type-Detection Logic for Rich Semantic Data Types using Open-source Code" (see bibliography [4]). In this paper, a method called "AutoType" is described. This method primarily uses open-source code and provides a function that expects two inputs from the user. The first input is a set of positive examples for a target data type, and the second input is a search keyword, which represents the target data type. AutoType then synthesizes type-detection functions based on these inputs, which can be used to identify the target data type in datasets

2 Definitions

Semantic data types refer to categories of data with a specific meaning or interpretation, beyond their raw values. An Example would be a Date. This type would contain all values which would represent a Date such as $\{01.01.20,\ 01/1/2020,\ 01.\ Nov.\ 2020, \dots\}$

Definition 1 (Binary Semantic Data Type) *A binary semantic data type is a semantic datatype which can be divided into two categories. These two categories should be as contrasting as possible. A binary semantic data type is a set of multiple tuples with two elements, where the tuples represent all variations of the two contrasting categories of the top level semantic data type*
Examples: $Gender = \{(male, female), (m, f), (Male, Female), \dots\}$, $Negations = \{(X, not\ X), (owned\ by\ X, not\ owned\ by\ X), \dots\}$, $Binary\ Scales = \{(>X, <=X), (18+, Juniors), \dots\}$, ...

Definition 2 (Binary Score) *The binary score determines the ranking of a candidate for a binary data type representative. If there are two candidates for binary data type representatives, the candidate with a higher binary score is more likely to be a binary data type representative. It is important to note that the binary score should not express the probability itself of being a representative. Furthermore, the two candidates do not necessarily have to share the same semantic data type either. The binary score simply indicates the order in which candidates can be sorted from best to worst suited.*

Definition 3 (Binary Rank) *The rank represents the probability of a candidate being a binary semantic data type representative in a given data set. If we have generated a list of candidates sorted in descending order by their score, the rank is calculated as the position of the candidate divided by the total number of candidates. The rank describes how likely it is that a candidate is a binary semantic data type representative.*

3 Score Calculation

In this chapter, we will discuss how to obtain candidates for our score calculation. Then, we will discuss heuristics for calculating our binary score. Finally, we will show how to calculate the binary rank.

3.0.1 Obtaining Candidates For Binary Semantic Data Types

Candidates are considered as tuple elements of a binary semantic data type. (see definition 1) If we have a collection of relational data tables, we can determine all candidates for binary semantic data types in linear time n, where n is the total number of entries across all relational tables. To do this, we iterate through each attribute of every table and examine all elements of the attribute. To determine if an attribute is a suitable candidate, we take all elements of that attribute and insert them into a set to remove duplicates. If this set contains only two elements, we save this set as a candidate.

3.0.2 Heuristics to evaluate Candidates

To calculate the score, we use various heuristics that are intended to provide insights into whether the score will increase or decrease. The following section will explain these heuristics.

3.0.2.1 Number Of Occurrences

The most obvious method to evaluate a candidates score is to examine how frequently the candidate appears in all relational tables. The more often it appears, the higher its score will be. The advantage here is clearly, that frequently used candidates for binary data types such as ("true", "false") or ("0", "1") receive a higher score. However, the disadvantage lies with candidates that are obviously binary but do not occur frequently in the relational tables. Examples of these might be ("Logged Out", "Logged In") or ("18+", "Juniors").

3.0.2.2 Equal Data Type

Here, we investigate whether the data types of the elements in the candidate are equal. Specifically, we examine the data types Integer, Float, and String. Equal data types in the candidate are rewarded with an increase of the binary score. Data belonging to the same semantic data type are likely to share the same primitive data types. While one could interpret a candidate $k = (0, \text{"true"})$ as a semantic truth-value data type, such representations are unlikely to be the sole truth-values in relational tables. This heuristic would be effective in scenarios like ("false", "true"), ("Logged Out", "Logged In"), (-12342, "Peter"), ...

3.0.2.3 Similarity

In this heuristic, we examine the similarity of elements within a candidate using a similarity measure. We use the token-based Jaccard-Similarity with a token length of one. We choose a small token length because short representations of binary semantic data types, such as (">0", "<=0") or ("1", "-1"), would not completely reduce to 0. Moreover, small token lengths also perform well for longer candidates, such as ("Logged Out", "Logged In") or ("Owned By", "Not Owned By"). Our observation shows that many candidates for binary semantic data types have similar elements, but there are also many where this is not the case. Therefore, this heuristic should be used with caution. Poor examples would be ("false", "true") or ("0", "1").

3.0.2.4 Multiple Occurrence of Candidate Elements

Another method to influence the score is to check if elements of one candidate exist in other candidates. The idea is to eliminate semantic data types that, may have only two representatives in one table, but have a different representatives in other tables. An example would be having a distinction of ("good", "bad") in one table, but having the distinction of ("good", "average") in another table. While ("good", "bad") can be interpreted as a binary semantic data type that expresses an evaluation, the probability of being a binary data type representative becomes lower when there is a third representative that is not completely contrasting to the other representatives. In this case, the semantic data type of evaluation seems to consist of the three elements "good", "average", and "bad".

One must exercise caution when applying this heuristic as it is particularly vulnerable to spelling errors. For instance, it is possible to encounter two identified candidates that are almost identical, such as ("false", "true") and ("FALSE", "true"). In such cases, the presence of the term "true" in both candidates may result in lower ratings for both candidates, despite the fact that they are similar in meaning and differ only in spelling. Hence, when applying this heuristic, it is important to consider the possibility of spelling variations and other similar sources of variation that may result in inaccurate evaluations.

Another drawback of this heuristic is its computational complexity. While other heuristics can operate linearly in the number of candidates, this method not only needs to require that all candidates have to be created before multiple occurrences can be identified, but also the worst-case scenario results in a quadratic run-time, which can be prohibitive for large data sets. Hence, when considering the use of this heuristic, it is important to take into account the computational resources required and the potential impact on run-time performance.

3.0.2.5 Using Non Binary Semantic Data Types To Eliminate Candidates

Binary semantic data types are, first and foremost, semantic data types. There are many semantic data types that are relatively easy to recognize, and have a very low probability of being used solely in a binary fashion in a large collection of data. Examples of such data types include dates, links, credit card numbers, and so on. Therefore, we create a naive identification method for such semantic data types in order to rank candidates representing such data types lower, or even to remove them immediately from the candidate

list. For example, links could be easily recognized by filtering for the keywords "www." or "http".

3.0.3 Calculating The Binary Score and Rank

First, we determine each candidate. We apply our heuristics to each candidate and save the results in a CSV table. The table scheme is:
(binary_candidate, count, data_types, jaccard_similarity, multiple_occurrence, final_score, rank)
In this case, binary_candidate is the candidate tuple itself. Count is an integer. Data_types is a tuple of the data types of the candidates. Jaccard_similarity is a float between 0 and 1, and multiple_occurrence is a Boolean. Final_score is the attribute where we later put our score and rank is the attribute where we will put our rank. final_score and rank are float values. For the score we need weightings for each heuristic to get proper results. In the algorithm below we are using a list named weighting with four float values to weigh our heuristics. To calculate the score we are using the following algorithm:

Algorithm 3.1: Binary Score Calculation

$s_1, s_2, s_3, s_4 = 0$;
for *c* ***in*** *candidates* **do**
    s_1 = c.count / c.MAX_COUNT $*$ weighting[0] ;
    if *c.data_types[0] == c.data_types[1]* **then**
        $s_2 = 0.01 *$ weighting[1] ;
    end
    s_3 = c.jaccard_similarity $*$ weighting[2] ;
    if *c.multiple_occurrence* **then**
        $s_4 = -0.01 *$ weighting[3] ;
    end
    c.final_score = $s_1 + s_2 + s_3 + s_4$;
end

To calculate our binary rank we use our scored list of candidates and sort them by their score. The rank is simply calculated by the candidates index divided by the total number of candidates.

4 Experiment

For our experiments we use the gittables-dataset. (see bibliography [2]). It has 1,018,141 relational tables and is 19.8GB large. We are using Python 3.10 and a Windows 11 computer with a AMD Ryzen 9 6900HS CPU and 32GB RAM. We calculated 1,236,740 possible candidates. Afterwards we calculated the necessary values for our heuristics "Number of Occurrence" (see 3.0.2.1), "Equal Data Type" (see 3.0.2.2) and "Similarity" (see 3.0.2.3) in linear time n, with n = total number of candidates. To calculate the candidate list with these three heuristics it took 3.4 hours. By removing duplicates we are left with 1,356,96 possible candidates. After that we calculated our values for the heuristic "Multiple Occurrence of Candidate Elements" (see 3.0.2.4) which took 0.51 hours. This takes so long because of the worst case run time of n^2. The score and rank calculation is fast again because of the linear time. Both together took only 6.21 seconds. candidates that have a score of 0 were removed from our candidates list. We were left with 70,845 candidates. The Score were calculated with the weighting [count:10, data_types:1, jaccard_similarity:0.1, multiple_occurrennce:0.1]

In our analysis we found out that the "count" and "data_types" factors are really good measures to determine the score. "jaccard_similarity" gave some good results in this data set but all in all the result were more bad than good. "Multiple_occurrence" was by far the worst. Not only did it provide many false results due to its susceptibility to typos, but it is also very slow to calculate due to its quadratic run-time. This heuristic definitely failed our test and should not be used in the future. Therefore its weighting is pretty low.

Furthermore, the filters for non-binary semantic data types (see 3.0.2.5) were applied to the evaluated data set. Semantic data types that were already classified as non-binary from the beginning received a score of 0.

To get an impression on how well our ranking is your can see the 50 best candidates in figure 4.1. The 50 worst candidates are in figure 4.2

Although some false results may be present in our top ranks, our approach has proven to be effective to a considerable degree. This is particularly evident in the lower ranks, where candidates with highly diverse semantic elements are primarily located.

binary_candidate	count	datatypes	jaccard_similarity	multiple_occurrence	final_score	rank
('comment', 'story')	316323	('string', 'string')	0.2222222222222	True	0.835935185185185	1
('1', '0')	111918	('integer', 'integer')	0	True	0.295591032741849	0.999985884478573
('A', 'B')	29815	('string', 'string')	0	True	0.0792957691452513	0.999971768957145
('True', 'False')	24619	('string', 'string')	0.125	True	0.0666488959333761	0.999957653435718
('2', '1')	21882	('integer', 'integer')	0	True	0.0583967724446215	0.999943537914291
('0.0', '1.0')	9301	('float', 'float')	0.6666666666667	True	0.0308084634482264	0.999929422392863
('2', '0')	9435	('integer', 'integer')	0	True	0.0256059225854585	0.999915306871436
('imports', 'exports')	7423	('string', 'string')	0.5555555555556	True	0.0249350629472617	0.999901191350008
('Y', 'N')	8990	('string', 'string')	0	True	0.0244335976728428	0.999887075828581
('"\\\\\\"", "\\\"DNP - Coach\\\\\\\\\'s Decision "\\\")	8558	('string', 'string')	0	True	0.0232955204543036	0.999872960307154
('1', '-1')	5016	('integer', 'integer')	0.5	True	0.0181310077041505	0.999858844785726
('0', '3')	5116	('integer', 'integer')	0	True	0.0142277848380716	0.999844729264299
('good', 'average')	4609	('string', 'string')	0.1428571428571	True	0.0140826009595677	0.999830613742872
('variety', 'species')	3766	('string', 'string')	0.2	True	0.0123379601968031	0.999816498221444
('3', '4')	4026	('integer', 'integer')	0	True	0.0113562474116647	0.999802382700017
('Men', 'Women')	2092	('string', 'string')	0.6	True	0.0112612443082967	0.99978826717859
('people', 'person')	2557	('string', 'string')	0.4285714285714	True	0.0110576865524585	0.999774151657162
('1.0', '-1.0')	1411	('float', 'float')	0.75	True	0.0107171920262938	0.999760036135735
('owned by a public authority', 'not owned by a public authority')	344	('string', 'string')	1	True	0.00998958000735535	0.999745920614307
('Yes', 'No')	3507	('string', 'string')	0	True	0.00998974086614	0.99973180509288
('101', '100')	302	('integer', 'integer')	1	True	0.00987893361110848	0.999717689571453
('model_purchase_requisition', 'model_purchase_requisition_line')	220	('string', 'string')	1	True	0.00966290969462648	0.999703574050025
('Static', 'Dynamic')	2328	('string', 'string')	0.3333333333333	True	0.00966074945546167	0.999689458528598
('0.0', '2.0')	1270	('float', 'float')	0.6666666666667	True	0.00965129182302056	0.999675343007171
('11.0', '10.0')	196	('float', 'float')	1	True	0.00959968318248541	0.999661227485743
('base.group_user', 'base.group_hr_user')	452	('string', 'string')	0.9166666666667	True	0.00957965486754573	0.999647111964316
('10', '100')	179	('integer', 'integer')	1	True	0.00955489773638549	0.999632996442889
('base.group_sale_manager', 'base.group_sale_salesman')	116	('string', 'string')	1	True	0.00938892814201518	0.999618880921461
('access_sale_order_line_manufacturing_user', 'access_sale_order_manufacturing_user')	111	('string', 'string')	1	True	0.00937575595198579	0.999604765400034
('sale.order manufacturing.user', 'sale.order.line manufacturing.user')	111	('string', 'string')	1	True	0.00937575595198579	0.999590649878607
('app-supplychain-config-form', 'supply-chain-config-form')	34	('string', 'string')	1	True	0.00917290422553319	0.999576534357179
('Method_Citation,Hypothesis_Citation,Aim_Citation,Implication_Citation', 'Method_Citation,Hypothesis_Citation,Implication_Citation')	34	('string', 'string')	1	True	0.00917290422553319	0.999562418835752
('1', '11')	31	('integer', 'integer')	1	True	0.00916500091151555	0.999548303314324
('13.043478260869565', '8.695652173913043')	29	('float', 'float')	1	True	0.0091597320355038	0.999534187792897
('Not owned by a public authority', 'Owned by a public authority')	25	('string', 'string')	1	True	0.00914919428348028	0.99952007227147
('com.axelor.apps.purchase.db.PurchaseRequest', 'com.axelor.apps.purchase.db.PurchaseRequestCreator')	21	('string', 'string')	1	True	0.00913865653145677	0.999505956750042
('model_base_action_rule_lead_test', 'model_base_action_rule')	17	('string', 'string')	1	True	0.00912811877943326	0.999491841228615
('sales_team.group_sale_manager', 'sales_team.group_sale_salesman')	17	('string', 'string')	1	True	0.00912811877943326	0.999477725707188
('nz_21310', 'nz_21330')	16	('string', 'string')	1	True	0.00912548434142738	0.99946361018576
('link_tracker.model_link_tracker_code', 'link_tracker.model_link_tracker_click')	16	('string', 'string')	1	True	0.00912548434142738	0.999449494664333
('access_pos_multi_session_sync_pos', 'access_pos_multi_session_sync_multi_session')	16	('string', 'string')	1	True	0.00912548434142738	0.999435379142906
('perm.suppliermanagement.PurchaseOrderSupplierLine.rwc', 'perm.suppliermanagement.PurchaseOrderSupplierLine.rwcde')	15	('string', 'string')	1	True	0.0091228499034215	0.999421263621478
('2.5944828711436', '3.5944828711436')	14	('float', 'float')	1	True	0.00912021546541563	0.999407148100051
('perm.client.portal.OnlinePaymentMethod;rwcde', 'perm.client.portal.OnlinePaymentMethod;rwc')	14	('string', 'string')	1	True	0.00912021546541563	0.999393032578623
('1.0', '0.1')	12	('float', 'float')	1	True	0.00911494658940387	0.999378917057196
('asset.model_asset_asset', 'asset.model_asset_state')	12	('string', 'string')	1	True	0.00911494658940387	0.999364801535769
('-84.5351 29.0149 100000 2', '-84.5351 29.0149 100000 1')	12	('string', 'string')	1	True	0.00911494658940387	0.999350686014341
('Undergraduate Student', 'Graduate Student')	11	('string', 'string')	1	True	0.00911231215139799	0.999336570492914
('F', 'fF')	11	('string', 'string')	1	True	0.00911231215139799	0.999322454971487

Figure 4.1: Best 50 Candidates For Binary Semantic Data Types

binary_candidate	count	datatypes	jaccard_similarity	multiple_occurrence	final_score	rank
('Default', '0')	1	('string', 'integer')	0	True	-8.0698895327455E-05	0.000691660549941
('3', 'None')	1	('integer', 'string')	0	True	-8.0698895327455E-05	0.000677545028513
('1', 'None')	1	('integer', 'string')	0	True	-8.0698895327455E-05	0.000663429507086
('PRIMARY', '1')	1	('string', 'integer')	0	True	-8.0698895327455E-05	0.000649313985659
('two', '3')	1	('string', 'integer')	0	True	-8.0698895327455E-05	0.000635198464231
('\\\\\\\\\\\\\\\\\\N', '2')	1	('string', 'integer')	0	True	-8.0698895327455E-05	0.000621082942804
('X', '-26')	1	('string', 'integer')	0	True	-8.0698895327455E-05	0.000606967421377
('X', '-69')	1	('string', 'integer')	0	True	-8.0698895327455E-05	0.000592851899949
('X', '-25')	1	('string', 'integer')	0	True	-8.0698895327455E-05	0.000578736378522
('C', '-36')	1	('string', 'integer')	0	True	-8.0698895327455E-05	0.000564620857094
('-1', 'C')	1	('integer', 'string')	0	True	-8.0698895327455E-05	0.000550505335667
('-27', 'C')	1	('integer', 'string')	0	True	-8.0698895327455E-05	0.00053638981424
('C', '-15')	1	('string', 'integer')	0	True	-8.0698895327455E-05	0.000522274292812
('None', '26')	1	('string', 'integer')	0	True	-8.0698895327455E-05	0.000508158771385
('4', 'None')	1	('integer', 'string')	0	True	-8.0698895327455E-05	0.000494043249958
('1', 'long')	1	('integer', 'string')	0	True	-8.0698895327455E-05	0.00047992772853
('1.', 'm')	1	('float', 'string')	0	True	-8.0698895327455E-05	0.000465812207103
('99.', 'm')	1	('float', 'string')	0	True	-8.0698895327455E-05	0.000451696685676
('cas1|cas3|cas4|cas8b', '0')	1	('string', 'integer')	0	True	-8.0698895327455E-05	0.000437581164248
('TypeI', '0')	1	('string', 'integer')	0	True	-8.0698895327455E-05	0.000423465642821
('TypeI-B', '0')	1	('string', 'integer')	0	True	-8.0698895327455E-05	0.000409350121393
('1', 'Alpha.0')	1	('integer', 'string')	0	True	-8.0698895327455E-05	0.000395234599966
('y', '0')	1	('string', 'integer')	0	True	-8.0698895327455E-05	0.000381119078539
('TSTOP', '23103')	1	('string', 'integer')	0	True	-8.0698895327455E-05	0.000367003557111
('estimated effort', '0')	1	('string', 'integer')	0	True	-8.0698895327455E-05	0.000352888035684
('99.6', '0')	1	('float', 'integer')	0	True	-8.0698895327455E-05	0.000338772514257
('0.0', 'NAN')	1	('float', 'string')	0	True	-8.0698895327455E-05	0.000324656992829
('3', 'N')	1	('integer', 'string')	0	True	-8.0698895327455E-05	0.000310541471402
('N', '2.5')	1	('string', 'float')	0	True	-8.0698895327455E-05	0.000296425949975
('77', 'See End Item P5A')	1	('integer', 'string')	0	True	-8.0698895327455E-05	0.000282310428547
('1', 'This is a simple application of the XOR rule do not override anything')	1	('integer', 'string')	0	True	-8.0698895327455E-05	0.00026819490712
('0', 'CAP_DEFAULT_NOT_INTERESTED')	1	('integer', 'string')	0	True	-8.0698895327455E-05	0.000254079385693
('1', 'YELLOW')	1	('integer', 'string')	0	True	-8.0698895327455E-05	0.000239963864265
('10', 'abc d e')	1	('integer', 'string')	0	True	-8.0698895327455E-05	0.000225848342838
('None', '0.25')	1	('string', 'float')	0	True	-8.0698895327455E-05	0.00021173282141
('Sports:', '25')	1	('string', 'integer')	0	True	-8.0698895327455E-05	0.000197617299983
('1', 'Ncc 2')	1	('integer', 'string')	0	True	-8.0698895327455E-05	0.000183501778556
('0', 'Criterium')	1	('integer', 'string')	0	True	-8.0698895327455E-05	0.000169386257128
('?', '7')	1	('string', 'integer')	0	True	-8.0698895327455E-05	0.000155270735701
('1', 'true')	1	('integer', 'string')	0	True	-8.0698895327455E-05	0.000141155214274
('0', 'INACTIVE')	1	('integer', 'string')	0	True	-8.0698895327455E-05	0.000127039692846
('-1000', 'false')	1	('integer', 'string')	0	True	-8.0698895327455E-05	0.000112924171419
('TYPING_DURATION', '0')	1	('string', 'integer')	0	True	-8.0698895327455E-05	9.88086499915E-05
('1', ']')	1	('integer', 'string')	0	True	-8.0698895327455E-05	8.46931285642E-05
('>', '1')	1	('string', 'integer')	0	True	-8.0698895327455E-05	7.05776071368E-05
('2', '>')	1	('integer', 'string')	0	True	-8.0698895327455E-05	5.64620857094E-05
('-', '0.0076')	1	('string', 'float')	0	True	-8.0698895327455E-05	4.23465642821E-05
('3600', 'true')	1	('integer', 'string')	0	True	-8.0698895327455E-05	2.82310428547E-05
('123', 'Men 4-5')	1	('integer', 'string')	0	True	-8.0698895327455E-05	1.41155214274E-05

Figure 4.2: Worst 50 Candidates For Binary Semantic Data Types

5 Summery

We identified binary semantic data type representatives in a data set of 1,018,141 relational tables using a ranking system. We devised various heuristics to evaluate candidates for binary semantic data type representatives. Our method has room for improvement in terms of accuracy, but it is certainly very fast. Even a large data set with 1,018,141 tables can be analyzed in under 4 hours. Finding candidates is the longest process in our pipeline, but it is also parallelizable. With parallelization, a much faster analysis is possible. Our algorithm is not as accurate as an approach with a neural network could be, but we save the time required for training and preparing the data sets for the learning process. In future research, additional heuristics for identification could be possible to achieve an even more accurate result. All in all, this work already shows very positive results.

Bibliography

[1] Madelon Hulsebos, Kevin Hu, Michiel Bakker, Emanuel Zgraggen, Arvind Satyanarayan, Tim Kraska, Çagatay Demiralp, and César Hidalgo. "Sherlock: A Deep Learning Approach to Semantic Data Type Detection". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*. KDD '19. Anchorage, AK, USA: Association for Computing Machinery, 2019, 1500–1508. ISBN: 9781450362016. DOI: 10.1145/3292500.3330993. URL: https://doi.org/10.1145/3292500.3330993.

[2] "Hulsebos, Gittables: A large-scale corpus of relational tables, 2023." In: (). URL: https://gittables.github.io.

[3] Subhadip Maji, Swapna Sourav Rout, and Sudeep Choudhary. "DCoM: A Deep Column Mapper for Semantic Data Type Detection". In: *CoRR* abs/2106.12871 (2021). arXiv: 2106.12871. URL: https://arxiv.org/abs/2106.12871.

[4] Cong Yan and Yeye He. "Synthesizing Type-Detection Logic for Rich Semantic Data Types Using Open-Source Code". In: *Proceedings of the 2018 International Conference on Management of Data*. SIGMOD '18. Houston, TX, USA: Association for Computing Machinery, 2018, 35–50. ISBN: 9781450347037. DOI: 10.1145/3183713.3196888. URL: https://doi.org/10.1145/3183713.3196888.

Eidesstattliche Erklärung

Hiermit versichere ich Kevin Gieswein, dass meine "Lakehouse – Data Integration: Data Profiling – Binary Datatype Analysis " selbständig verfasst wurde und dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt wurden. Diese Aussage trifft auch für alle Implementierungen und Dokumentationen im Rahmen dieses Projektes zu.

Marburg, den 24. März 2023,

(Kevin Gieswein)