BUILDING AN OPEN SOURCE DATA PROFILING TOOL

DEPENDENCIES: DETECTION OF UNIQUE COLUMN COMBINATIONS IN A DATASET USING HCA & GORDIAN ALGORITHMS- AN ANALYSIS

SNEHA DAS

UNIVERSITY OF CALIFORNIA, SANTA CRUZ

TABLE OF CONTENTS

BACKGROUND

Data quality and integrity are of utmost importance to businesses. Despite best efforts, gremlins inevitably find their way into systems with the end result being poor data quality which has a host of negative consequences. A few concerns that come up are whether the data is of sufficient quality to support the different business purposes for which it is being used or would there be any specific issues within the data decreasing its suitability for these business purposes. With the myriad of ways that data is captured like through online transactions, manual screen entry, spreadsheet uploads, direct database changes, there are many opportunities for flawed data to enter systems. Hence there is a need to refine or clean data before using it.

Data Profiling is a systematic analysis of the contents of data. It is the process of examining the data available in an existing data source (e.g. a database or a file) and collecting statistics and information known as metadata about that data. This metadata retrieved can be used to find out if existing data can easily be used for other purposes.

Different kind of analysis can be performed on data. A few kinds are Completeness Analysis to check how often is a given attribute is populated, versus blank or null; Uniqueness Analysis to check how many distinct values are found for a given attribute across all records and whether there are duplicates; Value Distribution Analysis to check the distribution of records across different values for a given attribute; Range Analysis to find the minimum, maximum, average and median values found for a given attribute and Pattern Analysis to discover the formats found for a given attribute, and the distribution of records across these formats. Hence data profiling can help in quickly and thoroughly unveiling the true content and structure of data.

Data profiling tools provide a common repository for storing data profile results and other key metadata such as notes made during analysis. These tools or data profiling in general face mainly three challenges which are managing the input, performing the necessary computations and managing/interpreting the output results. The first challenge refers to the problem of defining the expected outcome, or specifically determining which profiling tasks should be performed on which parts of the input data. The second challenge addresses the main issue in data profiling which is the computational complexity of the data profiling algorithms used. The third challenge refers to meaningfully interpret the metadata retrieved which is applicable only to data instances and cannot be used to derive schematic/semantic properties about the data with certainty.

Multi-Column Profiling is a particular classification of data profiling tasks that generalizes profiling tasks on multiple columns and identifies inter value dependencies and column similarities. Different types of multi-column profiling tasks would be identifying correlations between values through frequent patterns and association rules, clustering approaches to identify coherent subsets of data records and outliers, defining summaries and sketches of large datasets related to profiling values across columns. Such metadata retrieved can be used in various applications such as in data exploration and analytics.

A very common aim of data profiling is to find suitable primary key candidates for data given in a table. Hence discovering unique column combinations which is a set of values which uniquely identify rows is an important task in data profiling.

The main idea behind the project is to analyze different multi-column profiling algorithms on different datasets ,study the pros and cons of these algorithms discussed in the papers and eventually find suitable primary key candidates from unique column combinations generated by them. The discovery of unique column combinations which is a kind of dependency detection will be added as one of the functionalities of the open source data profiling tool developed out of the class.

INTRODUCTION

Dependencies are metadata that explain relationships between columns. Discovering such dependencies in a dataset have two sides to it.One being that many sets of combinations of columns have to be examined to check for dependencies and the second being that not all dependencies detected are meaningful.

Let R and S denote two relational schemata, with r and s denoting relational instances of R and S respectively. The solution set to the number of potential attributes in R can be exponential in number of attributes. This means that any dependency algorithm can have a worst case exponential time- complexity. There are two approaches to finding dependencies in a dataset. Column-based or top-down approaches start with small dependencies that is the number of attributes they reference and work onto larger dependencies, pruning candidates along the way. Row-based or bottom-up approaches attempt to avoid repeated scanning of the entire relation during candidate generation. Studies conducted on detecting dependencies have shown that these algorithms do not reduce the time complexity to any extent, but rather gave an idea about which algorithms worked better over what kind of datasets, like datasets more number of rows or wider tables. Finding unique column combinations in a dataset is one kind of dependency detection among other dependencies such as functional dependencies, inclusion dependencies and many others.

i. Unique column combinations and Keys:

   Given a relation R with instance r, a unique column combination (a "unique") is a set of columns X contained in R whose projection on r contains only unique value combinations. A set of columns X contained in R is a non-unique column combination (a "non-unique"), its projection on r contains at least one duplicate value combination.

   Each superset of a unique is also unique while each subset of a non-unique is also a non-unique. Therefore, discovering all uniques and non-uniques can be reduced to the discovery of minimal uniques and maximal non-uniques.

   Uniqueness is a necessary precondition for a key. A primary key is a minimal unique that is explicitly selected to be the unique record identifier while designing the table schema. Since the discovered uniqueness constraints are only valid for a relational instance at a specific point of time, we refer to uniques and non-uniques instead of keys and non-keys.

Consider the following example with attributes A, B, C:

A	B	C
a	1	s
b	4	s
c	4	z

Here Uniques are: {A, AB, AC, BC, ABC} while Non-Uniques are: {B, C}
For this example {A, BC} is a candidate for primary key in this relation as the other combinations are redundant. ({AB, AC, ABC})

Uniques help to understand the structure and semantic properties of tabular data. It is of high significance in many data management applications such as data modeling, query optimization, anomaly detection and indexing. Identification of uniques also play an important role in data mining. To create an algorithm for the discovery of all minimal uniques within a relational instance using only polynomial time in the number of columns and in the number of uniques is stated as an open problem in data mining by Mannila.

The problem of discovering a minimal unique of size $k \leq n$ in a data source however is NP-complete .To discover all minimal uniques and maximal non-uniques of a relational instance, in the worst case, all subsets of the given relation have to be considered , no matter the approach used (breadth- first or depth-first) or direction (bottom-up or top-down). To identify a column combination K of fixed size as a unique, all tuples t_i must be scanned. A scan has a runtime of O(n) in the number n of rows and to detect duplicate values, either a sort in O(nlogn) or a hashing algorithm that needs O(n) space can be used.

The different approaches to unique discovery are distinguished into two different classes:

a) Row-based algorithms are based on a row-by-row scan of the database for all column combinations.
b) Column-based algorithms, contains algorithms that check the uniqueness of a column combination on all rows at once. Such column combinations are generated iteratively and each of them is checked only once.

GORDIAN- A ROW BASED APPROACH

Row-based processing of a table for discovering uniques requires multiple runs over all column combinations as more and more rows are considered. This approach benefits from the intuition that non-uniques can be detected without considering all rows of a table. A recursive unique discovery algorithm that works this way is the Gordian algorithm.

The functioning of the algorithm can be mainly divided into three main parts:

The algorithm consists of three parts:

(i) Pre-organize table data in form of a prefix tree.

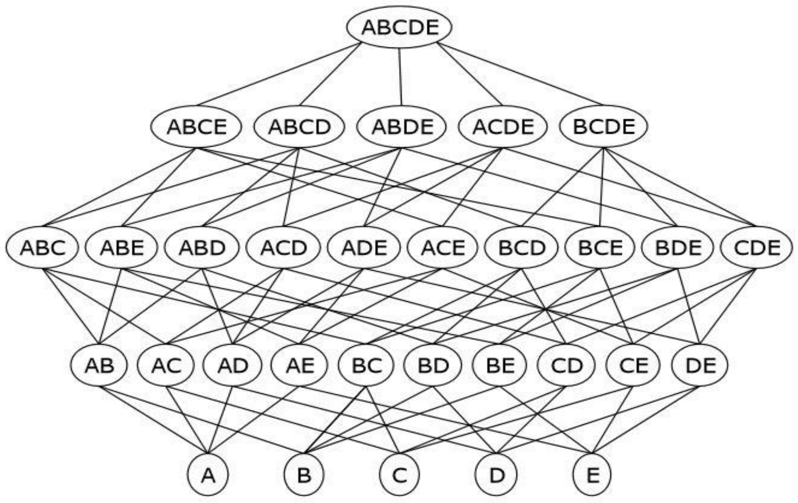


Figure 1

(ii) Find maximal non-uniques by traversing the prefix tree.

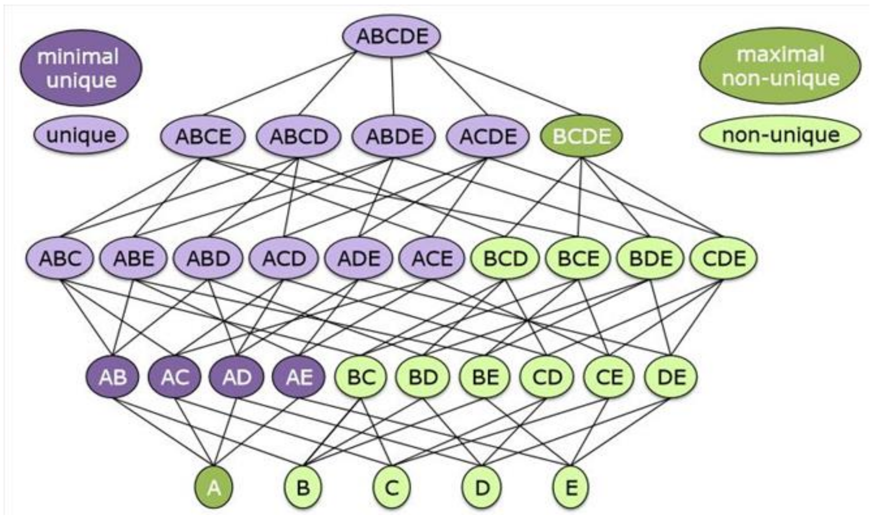


Figure 2

(iii) Compute minimal uniques from maximal non-uniques. The prefix tree has to be stored in main memory. Each level of the tree represents one column of the table whereas each branch stands for one distinct tuple. Non-unique discovery is performed by a depth-first traversal of the tree for discovering maximum repeated branches, which constitute maximal non-uniques.

After the discovery of all maximal non-uniques, Gordian computes all minimal uniques by generating minimal combinations that are not covered by any of the maximal non-uniques. This algorithm needs only quadratic time in the number of minimal uniques with a Zipfian distribution of values, but the presented algorithm implies cubic runtime. In the worst case, it has a worst case running time as per experimental results. The generation of minimal uniques from maximal non-uniques marks a serious bottleneck of the algorithm in case of large numbers of maximal non-uniques.

Experiments pose that in most cases the unique generation dominates the entire algorithm. The approach is limited by the available main memory and must be used on samples for approximate solutions when dealing with large data sets. Although data may be compressed because of the prefix structure of the tree, the amount of processed data may still be too large to be maintained in main memory. When different test datasets such as ‘titanic.csv’,’sales.csv’ were tested on the algorithm, the algorithm ran into the issue mentioned previously. Another issue faced was that the prefix tree was larger than the amount of physical memory, depending on the size of the dataset. Gordian was implemented and found correctness in the small test datasets as well as a few toy examples that were created.

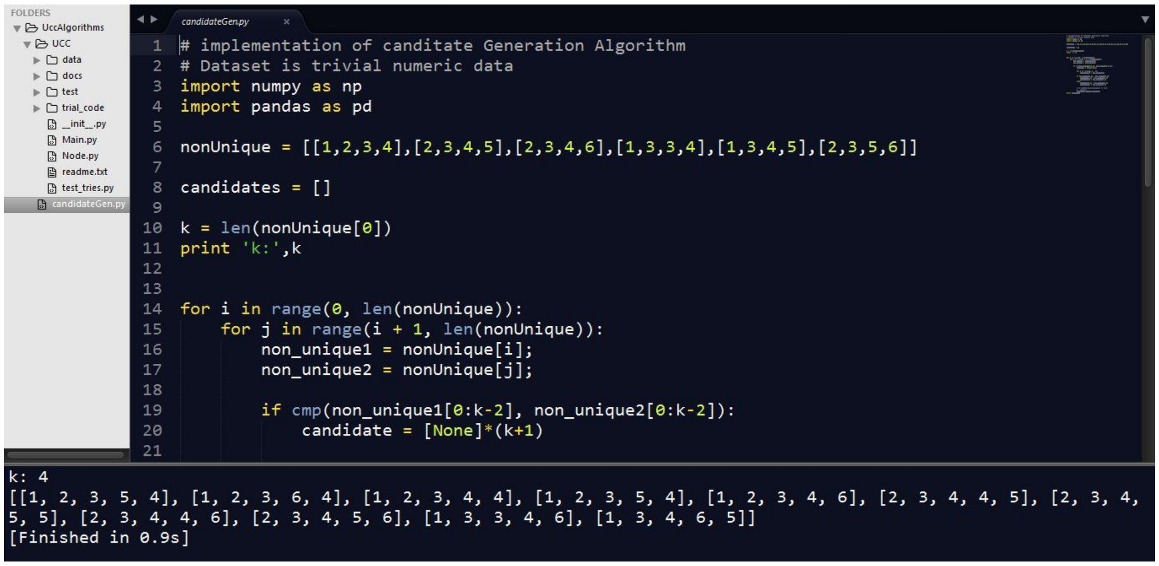
HCA- A COLUMN BASED APPROACH

The problem of finding minimal uniques is comparable to the problem of finding frequent item sets in data mining making use of the well-known Apriori approach, which is applicable for minimal unique discovery, working bottom-up as well as top-down. It generates all relevant column combinations of a certain size and verify all these combinations at once.

Bottom-up unique discovery mentioned here refers to the lattice of the schema R being traversed beginning with all 1 –combinations to the top of the lattice, which is the R combination. The prefixed number k of k-combination which indicates the size of the combination. The algorithm begins with checking the uniqueness of all individual columns. If a column is a unique, it will be added to the set of uniques, and if not it will be added to the list of 1-non-uniques. The next iteration steps are based on candidate generation. A k-candidate is a potential k-unique. In other words, all possible k –candidates need to be checked for uniqueness. Effective candidate generation leads to the reduction of the number of uniqueness verifications by excluding apriori known uniques and non-uniques.

The Histogram-Count-based Apriori Algorithm (HCA), is an optimized bottom-up algorithm. It outperforms Gordian given a threshold of minimum average distinctness. It applies an efficient candidate generation by performing statistical pruning at every level, by considering column statistics and use of FDs.

Candidate generation is an important part of both the top-down and bottom up approaches. More the number of candidates are pruned apriori, fewer the number of uniqueness checks that have to be performed and better runtime can be achieved. It benefits from all optimizations of the classical apriori candidate generation in the context of mining association rules, so that repeated and redundant candidates are pruned apriori. The intuition behind the candidate generation is that a (k+1)-unique can only be a minimal if the set of k -non-uniques contains all of the candidate's k-subsets. A minimality check is much cheaper than a verification step, hence this step is already performed within candidate generation. A simple candidate generation algorithm discussed, was implemented on an input dataset.



```python
# implementation of canditate Generation Algorithm
# Dataset is trivial numeric data
import numpy as np
import pandas as pd

nonUnique = [[1,2,3,4],[2,3,4,5],[2,3,4,6],[1,3,3,4],[1,3,4,5],[2,3,5,6]]

candidates = []

k = len(nonUnique[0])
print 'k:',k


for i in range(0, len(nonUnique)):
    for j in range(i + 1, len(nonUnique)):
        non_unique1 = nonUnique[i];
        non_unique2 = nonUnique[j];

        if cmp(non_unique1[0:k-2], non_unique2[0:k-2]):
            candidate = [None]*(k+1)

```

```
k: 4
[[1, 2, 3, 5, 4], [1, 2, 3, 6, 4], [1, 2, 3, 4, 4], [1, 2, 3, 5, 4], [1, 2, 3, 4, 6], [2, 3, 4, 4, 5], [2, 3, 4,
5, 5], [2, 3, 4, 4, 6], [2, 3, 4, 5, 6], [1, 3, 3, 4, 6], [1, 3, 4, 6, 5]]
[Finished in 0.9s]
```

Figure 3

Real-world data contains semantic relations between column entries, such as correlations and functional dependencies and knowing such relations and dependencies can help in reducing the number of uniqueness checks however unfortunately, these dependencies are usually not known.

This algorithm is based on a hybrid verification scan which retrieves either the number of distinct values and the histogram of value frequencies of a combination or only the number of distinct values. A candidate is a unique if it contains as many distinct values as there are tuples in the table or if all value frequencies are 1.

One advantage of the HCA approach discussed is the apriori identification of non-uniqueness of a k-candidate by considering the value frequencies of its combined (k-1) non-unique subsets. The union of two non-unique combinations cannot be a unique if the product of the count-distinct values of these combinations is below the instance cardinality. It is more than enough to identify a value within one of the (k-1) non-uniques that has a higher frequency than the number of distinct values within the other (k-1) non-uniques.

The proposed algorithm has the following drawbacks:

(i)Such a cluster of value frequencies appear only in early passes of the algorithm.

(ii)Histograms must be stored in memory.

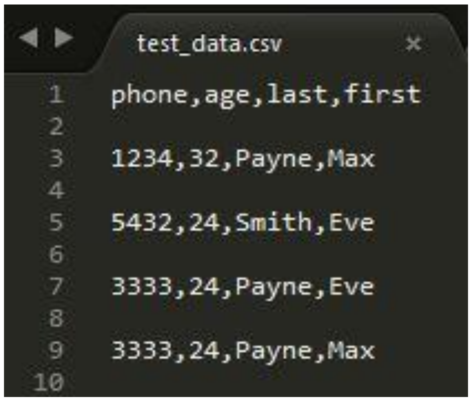
It works slowly on datasets with many non-uniques and works better on datasets with many minimal uniques. An important observation made was that it works better for datasets with larger number of rows.

EVALUATION AND RESULTS

The algorithms were tested on a synthetic test dataset ‘test_csv’ and real data. The algorithms were compared with regard to increasing number of rows, columns and average distinctness. Other important parameters for the algorithms are the number of uniques and their average size. Unfortunately, both values are only available after a successful completion of one algorithm. The generation of random data with specific number and size of uniques is almost as hard as the problem of discovering the minimal uniques and is an important challenge for future work. However these values were also analyzed when looking at the runtime of each algorithm.

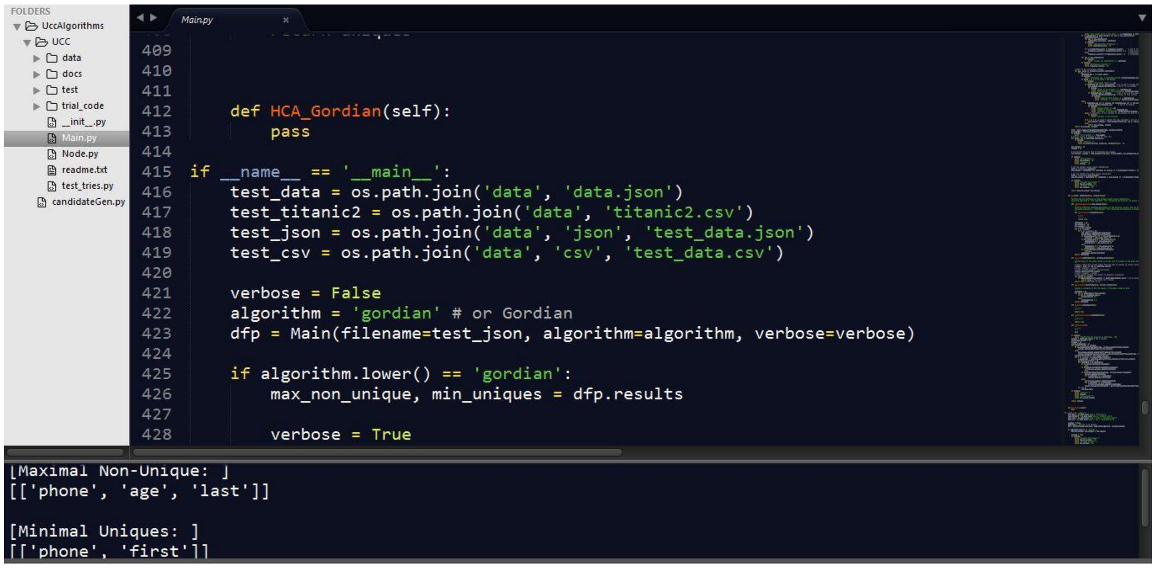
A few difficulties faced were in deciding on how to implement these algorithms. For example, a JSON file may have some header information that isn't relevant. The 'data.json' file mentioned in the package has @context, @type, conformsTo, dataset, and describedBy as keys. Obviously in this case, the "dataset" key is the one that contains the data, but deciding to make that dataset key known for future arbitrary json files if we assume not all files have keys or headers in the data was a challenge. Hence the information is fed into a data processing script to maintain robustness in the script.CSV files were much easier as the header need not be present.

Real world data may differ in its nature from domain to domain. All algorithms show strengths and weaknesses for different value distributions, size and number of uniques. Efficient candidate generation leads to remarkable runtime improvement of the bottom-up algorithms. A synthetic test dataset that was considered is ‘test_data.csv’.



```
phone,age,last,first

1234,32,Payne,Max

5432,24,Smith,Eve

3333,24,Payne,Eve

3333,24,Payne,Max

```

Figure 4



```python


    def HCA_Gordian(self):
        pass

if __name__ == '__main__':
    test_data = os.path.join('data', 'data.json')
    test_titanic2 = os.path.join('data', 'titanic2.csv')
    test_json = os.path.join('data', 'json', 'test_data.json')
    test_csv = os.path.join('data', 'csv', 'test_data.csv')

    verbose = False
    algorithm = 'gordian' # or Gordian
    dfp = Main(filename=test_json, algorithm=algorithm, verbose=verbose)

    if algorithm.lower() == 'gordian':
        max_non_unique, min_uniques = dfp.results

        verbose = True
```

```
[Maximal Non-Unique: ]
[['phone', 'age', 'last']]

[Minimal Uniques: ]
[['phone', 'first']]
```

Figure 5

JSON data was formatted as CSV data so that only one algorithm had to be written to solve both types of files. This works fine for JSON data that has all columns present. In the presence of missing columns, a separate algorithm had to be used to handle each type of file separately. Set notation is used to include each new key as it was encountered in each row in the code. This allows the set of headers to be distinct and have unique values, while maintaining a standard of having a value for each header key. A lot of the rows will have NaN values, but these are easy enough to deal with in Pandas/NumPy.

Both algorithms were implemented and tested. Gordian found correctness in the test datasets as well as in a few test datasets that were created. However, on larger datasets, running the algorithm was a challenge. The generation of minimal uniques from maximal non-uniques marks a serious bottleneck of the algorithm in case of large numbers of maximal non-uniques. Experiments showed that in most cases the unique generation dominates the entire algorithm. The algorithm gets stuck while running it on large datasets due to the high time complexity. Furthermore, the approach is limited by the available main memory and must be used on samples for approximate solutions when dealing with large data sets. Although data may be compressed because of the prefix structure of the tree, the amount of processed data may still be too large to be maintained in main memory.HCA works correctly overall but again the algorithm itself has limitations.

CONCLUSION AND FUTURE WORK

The concepts of uniques and non-uniques, the effects of their size and numbers, the strengths and weaknesses of existing approaches were all analyzed as part of this project. The new bottom-up algorithm HCA, which benefits from apriori candidate generation and data- and statistic-oriented pruning possibilities and the Gordian algorithm were analyzed. The HCA approaches are much more memory efficient than Gordian.

As future work, the non-unique discovery of Gordian can be interlaced with the candidate generation of HCA. The unique-generation part of the Gordian algorithm is inefficient if the number of discovered non-uniques is high. By profiling the runtime of Gordian the unique generation can be identified as the bottleneck. At the same time the non-unique discovery consumed only a fraction of the runtime. The non-unique discovery of Gordian can be performed on a smaller sample of the table and executing HCA on the entire table. Non-uniques discovered within a sample of a relational instance are also non-uniques for the complete instance and can be used for pruning candidates during the HCA part of the algorithm hence making it possible to smooth the worst case of the bottom-up algorithm by skipping non-uniques identified by Gordian and simultaneously to avoid Gordian's bottleneck of unique generation .

An important application of this project which can be used for further research is approximate unique discovery. Another open issue discussed is the need for a flexible and efficient data generator that should be able to generate a table that contains a fixed number of uniques of a certain size and holds specific value distributions for all columns making it possible to evaluate and benchmark algorithms for special cases .Recent proposals for column stores call for unique discovery solutions that benefit from features of a column-based DBMS. As HCA is a statistics-based approach, it allows further optimizations based on statistics-driven heuristics for approximate solutions and hence can compete as a suitable candidate.

LIST OF FIGURES

BIBLIOGRAPHY

1. Ziawasch Abedjan, F. N. (n.d.). Advancing the discovery of unique column combinations.
2. Sismanis, Y., Brown, P., Haas, P. J., & Reinwald, B. (n.d.). Gordian-Efficient and Scalable discovery of Composite Keys.
3. Srikant, R. A. (1994). Fast algorithms for mining association rules in large databases. *VLDB*, 487-499.
4. Ziawasch Abedjan, F. N. (n.d.). Scalable Discovery of unique column combinations.
5. Ziawasch Abedjan, L. G. (n.d.). Profiling Relational Data –a survey ., (pp. 1-14).