International Journal of Engineering Research and Development
e- ISSN: 2278-067X, p-ISSN: 2278-800X, www.ijerd.com
Volume 21, Issue 11 (November 2025), PP 01-07

Cleaning and Preprocessing of Data in Data Warehouses

L.S. Lazuta[1], A.A. Karpuk[2*]
[1]*Departmentof Telecommunication Systems, Faculty of Telecommunication*
[2]*Department of Telecommunication Network SoftwareSystems, Faculty of Telecommunication*
Belarusian State Academy of Communications,Minsk, BELARUS
**Corresponding Author*

ABSTRACT
This article provides an overview of the main methods for cleaning and preprocessing data in Big Data warehouses. It addresses issues such as removing duplicates and inconsistencies, handling missing or incomplete data, and normalizing and standardizing data. It demonstrates the use of rule-based, statistical, machine learning, and hybrid methods for cleaning and preprocessing data. Algorithms for finding functional dependencies between data in data warehouses, including those based on attribute lattice analysis, those based on table row analysis, and hybrid algorithms, are discussed.
Keywords*: Big Data systems, data warehouse, data cleaning, data preprocessing, data normalization, functional dependencies between data*

Date of Submission: 01-11-2025 Date of acceptance: 08-11-2025

I. INTRODUCTION

Modern Big Data systems can have the architecture of a Relational Data Warehouse, Data Lake, Modern Data Warehouse, Data Fabric, Data Lakehouse and Data Mesh [1]. In all of the listed architectures, except for the Lake Data architecture, the tasks of data cleaning and pre-processing are solved at one or more stages of data processing. Data cleaning includes identifying and correcting errors, inconsistencies, and inaccuracies in a data set. Data pre-processing is aimed at transforming and organizing data into formats most suitable for analysis, which simplifies the application of algorithms for analyzing and using data. At this stage, the problem of identifying and searching for functional dependencies (FD) between data is also solved. The found FDs are then used to normalize data in data warehouses.

II. MAIN SECTION

2.1. Data cleaning and preprocessing methods

Data cleaning is a fundamental step in preparing raw data for analysis and ensuring its high quality, consistency and usefulness for subsequent decision-making tasks. It aims to improve the quality of data by removing errors that may distort the results of the analysis. Key tasks in the data cleaning process are removing duplicates and inconsistencies, handling missing or incomplete data, normalization and standardization of data.

Duplicates and inconsistencies in data sets can lead to overestimates or inaccurate conclusions. The cleaning process involves identifying and removing duplicate records and resolving inconsistencies in data records (such as differences in spelling, capitalization, or abbreviations of words). Ensuring consistency in a data set is essential for accurate analysis. Missing data is a common problem in large data sets. Effective handling of missing values is critical to avoiding errors in analysis. There are various methods for managing missing data, including imputation, deleting records with missing values, although this may result in data loss, and isolating missing data into a separate category for specialized analysis.

Data normalization and standardization are techniques used to adjust the scales of attribute values so that they can be compared or processed effectively. This is especially important when working with machine learning models, where some algorithms may perform better when input features are scaled equally. Normalization rescales the data to a range (e.g., 0 to 1) to ensure that all features contribute equally to the model. Standardization transforms the data so that the mean is zero and the standard deviation is 1, which is important for analysis algorithms that are sensitive to the scale of the input data.

Data preprocessing involves feature selection, feature extraction, scaling, encoding, and binning. Feature selection involves identifying the most relevant attributes from a dataset, while feature extraction involves creating new features from existing attributes. These tasks reduce the dimensionality of data, which improves the performance of data analysis and processing algorithms. Data scaling involves changing the scale of features to ensure that the range is uniform across attributes. This helps avoid biased results when distance-based algorithms or gradient-based methods analyze and process data. Data encoding involves converting

categorical data into numeric formats using methods such as one-hot encoding, label encoding, or binary encoding, which are required by many data analysis and processing algorithms. Data binning involves converting continuous feature values into discrete values or categories, which can help smooth out noisy data and identify patterns.

In modern Big Data systems, rule-based methods, statistical methods, machine learning (ML) methods, and hybrid methods can be used for data cleaning and preprocessing [2]. Rule-based methods rely on standardization of data formats, constraint-based checks, and domain-specific business rules. These methods typically use predefined rules or heuristics to identify and correct errors in data. These methods include checking whether data complies with a specific format (e.g., a date in the DD.MM.YYYY format), checking whether numeric data complies with a specified range (e.g., age from 0 to 120), checking the consistency between related features (e.g., matching country codes with the names of the corresponding countries).

Statistical data cleaning and preprocessing methods are based on statistical data analysis metrics and are primarily used to estimate missing data values, find outliers in data, normalize and standardize data, and scale data. For example, missing feature values can be replaced by mean or median values, outliers in the data can be found based on the mean and standard deviation using the three sigma rule or the Z-score method. If feature values do not follow a normal distribution, outliers can be found using the interquartile range method [3].

Machine learning methods for solving data cleaning and preprocessing problems have several advantages over traditional methods. Machine learning algorithms are capable of automatically processing huge amounts of data without the need for manual intervention, making them well suited for Big Data systems. The tasks of detecting outliers, anomalies, or duplicates can be automated using machine learning models. This reduces the need for human intervention and speeds up the cleaning process. Machine learning models can adapt to a wide range of data types and handle complex relationships and patterns between data that are difficult to capture using rule-based or statistical methods. Machine learning models can identify both subtle and complex errors or inconsistencies in data that traditional methods may miss, such as noise in sensor data or misclassification of categorical data.

Machine learning models can be trained to detect outliers or anomalies in large data sets. These models improve over time as they learn from new data. ML algorithms are able to predict and fill in missing values based on patterns in data, unlike traditional methods that often rely on simple techniques like calculating the average. ML models can learn to identify and remove duplicate records by recognizing patterns and similarities, even if they are not exact matches.

Machine learning algorithms have advanced capabilities for detecting complex patterns and errors in data. Unsupervised learning methods such as *k*-means clustering or DBSCAN can detect anomalies by grouping similar data points and identifying outliers as data points that do not fit into any cluster. Supervised learning methods such as decision trees, random forests, or support vector machines can be trained to classify data as valid or invalid based on labeled training data sets. Neural networks, especially autoencoders, can be used to detect anomalies by learning normal patterns in data and identifying outliers. Reinforcement learning can optimize the sequence of actions performed during the data cleaning process, such as which data points to inspect first or which methods to use for cleaning. Over time, reinforcement learning models improve their data cleaning workflows by learning the consequences of their actions, resulting in improved performance and a reduced need for manual corrections.

Many ML models, especially deep learning models, operate as black boxes, making it difficult to understand how they make decisions. This lack of transparency creates a problem when cleaning data, as it may not always be clear why a particular decision was made (for example, why certain data values were removed or changed). The inability to interpret the decisions made by an ML model can lead to problems when validating cleaned data, making it difficult to explain the process to stakeholders or ensure that the data meets regulatory or ethical standards. Methods such as SHAP (Shapley Additive Explanations) or LIME (Local Interpretable Model-Agnostic Explanations) can provide insight into how models make predictions. Additionally, simpler models such as decision trees or rule-based algorithms may be preferable in scenarios where interpretability is critical.

While ML can automate many aspects of data cleaning, human intervention and review are often still necessary to ensure the quality and correctness of the cleaned data. Automated systems can make mistakes, especially when the data is noisy, sparse, or contains ambiguities that ML models cannot resolve on their own. A hybrid approach that combines ML automation with occasional human review is often the best solution. For example, an ML model can be used to identify potential issues that can then be reviewed by a data expert. This balance helps ensure that automation is effective, while human review provides oversight and resolves ambiguities.

Future research in the area of using ML methods for data cleaning and preprocessing in Big Data systems should focus on developing deep learning methods, developing automated end-to-end ML pipelines,

and leveraging cloud-based data quality solutions. Deep learning as a form of ML is increasingly being used to solve complex data cleaning problems. With the growth of unstructured data such as images, text, and videos, deep learning models (e.g., convolutional neural networks for images, recurrent neural networks for sequences) show significant promise in identifying patterns and anomalies that traditional ML models may struggle to capture. Automated end-to-end data pipelines optimize the entire data lifecycle, from collection and cleaning to transformation and analysis, without the need for manual intervention. Cloud platforms increasingly incorporate ML algorithms to automate and scale data cleaning and preprocessing tasks. These platforms offer scalability, ease of use, and the ability to handle large volumes of data in distributed systems. Cloud service providers such as AWS, Google Cloud, and Microsoft Azure are integrating ML-based data cleansing capabilities into their platforms, offering data quality tools as part of their data management services [4].

2.2. Methods for identifying and searching for functional dependencies between data

Let$X \subseteq A$ be a subset of attributes (features) of the Big Data system, $Z \in A$ be some attribute. It is said that there is a FD $X \rightarrow Z$ if any combination of attribute values from X always corresponds to a unique value of attribute Z. Obviously, from $Z \in X$ it follows that $X \rightarrow Z$. Such a FD, in which the dependent attribute is part of the left part of the FD, is called trivial. In what follows, we will consider only non-trivial FDs between attributes. The structure of the FD on the set of attributes satisfies the axioms of W. Armstrong [5]. In the works on database design in the 80s of the last centuries, the structure of the FD on the set of attributes of the subject area was specified by postulating a certain set of FDs, which is called the system of generators of the FD structure. P. Chen proposed to describe the subject area when designing databases in the form of an entity-relationship diagram (ER diagram) [6]. From the ER diagram, it is possible to extract and include in the system of forming the FD structure the dependencies of all attributes of entities on primary and potential (alternative) keys of entities. It is impossible to obtain other non-trivial FDs between attributes from the ER diagram. A.A. Karpuk and V.M. Ostreiko proposed a method for constructing an extended ER diagram, which differs from P. Chen's diagram in that the entities of the subject area can have a complex hierarchical structure, and the diagram provides for the postulation of FDs and non-functional relationships between attributes that exist in isolation from the context [7]. From the extended ER diagram, it is possible to additionally extract and include in the system of forming the FD structure the dependencies of non-key attributes of complex objects on concatenated keys and FDs between attributes that exist in isolation from the context.

Research on finding FD between attributes was resumed at the beginning of the 21st century in connection with the emergence of the ontological approach to describing the domain and the development of Big Data systems. A.A. Karpuk developed a method for constructing a system of forming structures of FDs between attributes of a domain based on the ontology of the domain described in the OWL-2 language [8]. This method identifies FDs between attributes included in one class, included in a class and its subclass, included in functional relations between classes, included in the definition of functions specified on the attributes and classes of the ontology of the subject area.

A number of authors have proposed algorithms for searching for FDs between attributes in Big Data system tables obtained after cleaning and pre-processing of data. In general, this problem is NP-hard, since for a table containing m attributes and n rows, the computational complexity of the problem is estimated as$O(2^{m-2}n^2m^2)$ [9]. Existing algorithms for searching for FDs between attributes in tables can be divided into 3 classes: algorithms based on the analysis of the attribute lattice, algorithms based on the analysis of table rows, and hybrid algorithms.

An attribute lattice for a table containing m attributes is a directed graph consisting of $m + 1$ levels. At level 0, there is a single vertex of the graph, representing the empty set of attributes. At level 1, there are m vertices, each representing 1 attribute of the table. At level 2, there are $m(m-1)/2$ vertices, each representing a combination of two attributes of the table. In general, at level $1 \le k \le m$, there are $m!/k!(m-k)!$ vertices, each representing a combination of k attributes of the table. At level m, there is a single vertex representing the entire set of attributes of the table. In an attribute lattice, there exists a directed edge between vertices of adjacent levels, from a lower level to a higher level, if all attributes of a vertex from a lower level are included in the attributes of a vertex from a higher level. Each edge of the attribute lattice corresponds to a possible FD, the left part of which contains the attributes of a lower-level vertex, and the right part contains the attribute obtained by removing all attributes of a lower-level vertex from the attributes of a higher-level vertex. The left part of Figure 1 shows the attribute lattice for a table of four columns A, B, C, D. The vertices A from level 1 and AB from level 2 correspond to the possible FD $A \rightarrow B$, the vertices CD from level 2 and BCD from level 3 correspond to the possible FD $CD \rightarrow B$.

The idea of algorithms for searching for FDs between attributes in tables based on the analysis of the attribute lattice is to identify a possible FD, check the possible FD and classify it as is-FD or not-FD, and reduce the search area based on the obtained classification by removing some possible FDs or removing some lattice

vertices. If *X* and *Y* are 2 non-intersecting subsets of attributes, *A* and *B* are 2 attributes that do not belong to *X* and *Y*, then by virtue of W. Armstrong's axioms the following rules are true.

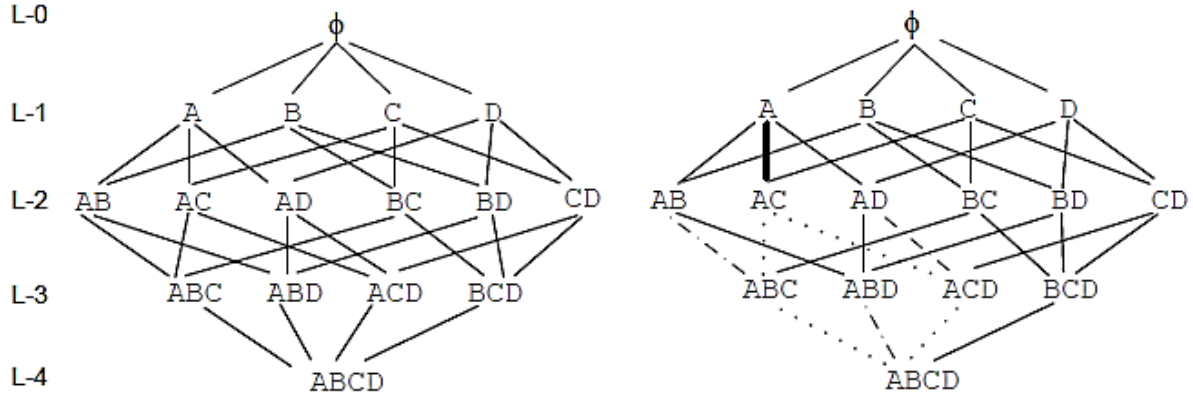

Figure 1: Lattice of attribute for a four-column table

1) If a possible FD $X \rightarrow A$ is classified as is-FD, then all FDs $XY \rightarrow A$ will also be classified as is-FD, and there is no need to check them, and the corresponding edges should be removed from the attribute lattice.

2) If a possible FD $X \rightarrow A$ is classified as is-FD, then instead of checking a possible FD $XAY \rightarrow B$, a possible FD $XY \rightarrow B$ should be checked, and the corresponding edges should be removed from the attribute lattice.

3) If *X* is the key of the table under consideration, i.e. for any attribute *Z* not belonging to *X*, a possible FD $X \rightarrow A$ is classified as is-FD, then all vertices containing all attributes from *X* should be removed from the attribute lattice.

For example, let's assume that in the attribute lattice shown in Figure 1, the possible FD $A \rightarrow C$ is classified as is-FD (the corresponding edge is highlighted in the right part of Figure 1). Then, according to rule 1, the edges between the vertices *AB* and *ABC*, *AD* and *ACD*, *ABD* and *ABCD* (these are the FDs $AB \rightarrow C$, $AD \rightarrow C$, $ABD \rightarrow C$, respectively) should be removed from the lattice. According to rule 2, the edges between the vertices *AC* and *ABC*, *AC* and *ACD*, *ABC* and *ABCD*, *ACD* and *ABCD* (these are the FDs $AC \rightarrow B$, $AC \rightarrow D$, $ABC \rightarrow D$, $ACD \rightarrow B$, respectively) should be removed from the lattice.

Using the above rules for reducing the search area in the attribute lattice, the algorithm for searching for FDs between attributes in the tables of Big Data systems, called the TANE algorithm [10], operates. In this algorithm, the lattice traversal is combined with its construction, starting from level 1, first to the right, and then down. By cutting off the edges of the lattice according to rules 1 and 2, the algorithm finds minimal FDs that do not contain unnecessary attributes in the left part. To classify possible FDs, the TANE algorithm divides the set of table rows into equivalence classes with matching attribute values of the left part of the possible FDs under consideration, and then checks the attribute values of the right parts of the possible FDs. The FUN algorithm proposed in [11] differs from the TANE algorithm in that it uses a stricter rule than rule 2 to cut off the edges of the lattice, thereby reducing the amount of computation.

In [12], an algorithm for finding FDs between attributes in Big Data system tables based on the analysis of the attribute lattice is proposed, called the FD_Mine algorithm. This algorithm also traverses the attribute lattice, starting from level 1, first to the right, and then downwards. To classify possible FDs the FD_Mine algorithm uses the same method as the TANE algorithm. Unlike the TANE and FUN algorithms, the FD_Mine algorithm uses an additional rule for cutting off the vertices and edges of the lattice, based on the equivalence classes of attribute subsets. Two sets of attributes are considered equivalent if they are functionally dependent on each other. After checking each level in the attribute lattice, the FD_Mine algorithm scans this level and the detected FDs for equivalence. The equivalence is checked by constructing the closure of each of the attribute subsets relative to the detected FDs. If equivalent subsets of attributes are found, the algorithm leaves the attributes of one of the equivalent subsets in the lattice, removing from the lattice the vertices containing the remaining equivalent subsets and the edges corresponding to them.

In [13], an algorithm for searching for FDs between attributes in Big Data system tables based on the analysis of the attribute lattice, called the DFD algorithm, is proposed. At the first step of the algorithm, each attribute is checked to see if it is a table key. If the attribute is a key, then the FDs are recorded, the left part of which consists of this attribute, and the right parts contain all the other attributes of the table, and this attribute is excluded from the lattice. Each remaining attribute is considered as a possible right part of some FDs. For each possible right part of the FD, the algorithm forms a set of sources for the left part of the FD and traverses the sources, choosing the next source according to certain rules, which in some situations provide for a random

choice. The algorithm classifies each source (lattice vertex) and the corresponding edge as a dependence, a minimum dependence, a candidate for a minimum dependence, a non-dependence, a maximum non-dependence, or a candidate for a maximum non-dependence.

Dependency and non-dependency represent the vertices of the left side of the FD that are neither minimal nor maximal, respectively. A minimal dependence or/and maximal non-dependency candidate is a combination of columns that can still be the left side of a minimal FD or maximal non-FD. In case the current combination of columns has already been visited at an earlier stage of the process, the DFD algorithm checks again only those combinations of columns that are classified as minimal dependence or maximal non-dependency candidates. In doing so, the algorithm checks whether the classification of the candidate can be changed or not. It may happen that after revisiting a minimal dependence candidate, all subsets of this vertex have been classified as non-dependency, making the candidate a minimal dependence. That is why the DFD algorithm stores the visited candidate vertices in a trace stack. The trace stack allows the algorithm to backtrack through the lattice, revisiting vertices that can eventually be classified at a later stage. In case the node has not been visited yet, the DFD algorithm checks whether the node is a superset or subset of a previously detected dependency or non-dependency and updates its classification accordingly. Otherwise, the algorithm determines what type of candidate the node combination represents. If a FD is detected, the node is classified as a candidate for minimal dependency, otherwise as a candidate for maximal non-dependency.

The algorithms for searching for FDs between attributes in tables based on the analysis of table rows include the FDEP [14], Dep-Miner [15], FastFDs [16], and FSC [17] algorithms. In [14], P.A. Flach and I. Savnik proposed three variants of the FDEP algorithm: a top-down analysis algorithm, a bidirectional analysis algorithm, and a bottom-up analysis algorithm. In a top-down analysis, the algorithm first determines for each attribute A the minimal subsets of attributes X for which the FD $X \rightarrow A$ is satisfied in the first two rows of the table. Then, the fulfillment of each found FD $X \rightarrow A$ is checked for the remaining rows of the table. If the FD $X \rightarrow A$ is not satisfied for the next row of the table, this FD is removed from the possible FDs under consideration. As a result, a set of minimal FDs is obtained for each attribute that are satisfied for all rows of the table.

In bidirectional analysis, all pairs of table rows are analyzed first. For each pair of rows, for each attribute A, the algorithm determines the maximum subsets of attributes Y for which the FD $Y \rightarrow A$ is not satisfied, i.e., it finds the maximum no-FDs. Based on the found maximum no-FDs, an irredundant negative coverage of the FD is constructed. The second step of the bidirectional analysis algorithm is similar to the top-down analysis algorithm, and differs from it in that the fulfillment of each FD $X \rightarrow A$ found for the first two rows is checked not by the remaining rows of the table, but by the constructed irredundant negative coverage of the FD. At the first step of the bottom-up analysis algorithm, the maximum no-FDs are also found and the irredundant negative coverage of the FD is constructed. At the second step of the algorithm, in one iteration, based on the irredundant negative coverage of the FD, a positive coverage of the FD is constructed, consisting of minimal FDs. Obviously, with a sufficiently large number of rows in the table, the performance of the bottom-up analysis algorithm will be higher than that of the bidirectional analysis algorithm.

In [15], S. Lopes et al. proposed the Dep-Miner algorithm, which derives all minimal FDs from sets of attributes that have the same values in certain rows of a table. These sets of attributes are called matched sets, and the sets of the remaining attributes are called complementary sets. The Dep-Miner algorithm consists of five stages. In stage 1, the algorithm computes a truncated partition of the table for each attribute in which each equivalence class of the attribute contains more than one row of the table. From the truncated partition of the table, a set of maximal equivalence classes for the table is constructed, which consists of those equivalence classes for all attributes that are not subsets of other equivalence classes. In stage 2 of the algorithm, the set of maximal equivalence classes for the table is used to construct matched sets of table attributes. A matched set of table attributes consists of one or more attributes that have the same values in two or more rows of the table. In step 3, the matched attribute sets of the table are transformed into maximal sets, i.e., sets of attributes that do not have a superset with the same values in two rows of the table. In step 4, the algorithm inverts the maximal matched attribute sets into complementary sets. In step 5, the algorithm computes all minimal FDs between attributes from the complementary attribute sets.

In [16], C. Wyss et al. proposed the FastFDs algorithm as an improvement of the Dep-Miner algorithm. The FastFDs algorithm also relies on the use of consistent attribute sets to infer the FDs between attributes. After computing the consistent attribute sets, the FastFDs algorithm follows a different strategy to infer the minimum FDs. Since maximizing the consistent attribute sets in step 3 of the Dep-Miner algorithm is an expensive operation, the FastFDs algorithm instead computes all additional attribute sets directly from the consistent sets. In step 4, the algorithm computes all minimum additional attribute sets for each attribute, based on which the algorithm finds all minimum FDs between attributes in step 5.

One of the latest algorithms for finding FDs between attributes in tables based on table row analysis is the FSC algorithm proposed by X. Wan et al. in 2024[17]. The two-step execution of the FSC algorithm relies

on a pre-computed auxiliary structure of comparable row pairs, which reflects the pairs of identifiers for rows that have at least one attribute with the same value. In Step 1, the FSC algorithm identifies the violated FDs and introduces selective comparison using the comparable row pairs to significantly reduce the required pairwise comparison. To handle low-cardinality attributes, a direct value and value combination compression strategy is developed. In Step 2, the FSC algorithm induces the desired FDs based on the results of Step 1. Extensive experimental results obtained on synthetic and real datasets show that the FSC algorithm can effectively detect FDs in Big Data tables.

Algorithms for searching for FDs between attributes in Big Data system tables based on attribute lattice analysis are critical to the growth of the number of attributes in tables and work well with a small number of attributes (no more than 20) for a large number of rows (up to hundreds of thousands). Algorithms for searching for FDs between attributes in tables based on table row analysis are critical to the growth of the number of rows and work well with a relatively small number of rows (no more than 1000) for a large number of attributes (up to several hundred). Hybrid algorithms for searching for FDs between attributes in Big Data system tables attempt to eliminate these shortcomings.

T. Papenbrock and F. Naumann developed a hybrid algorithm, HyFD [18], which combines attribute-efficient and row-efficient FD finding methods. In the first stage, HyFD extracts a small subset of the rows from the input and computes the FDs for this non-random sample. Since this stage uses only a subset of the rows, it is particularly attribute-efficient. The result is a set of FDs that are either valid or nearly valid with respect to the full input. In the second stage, HyFD checks the discovered FDs against the entire row set and finds those FDs that fail. This stage is row-efficient because it uses previously discovered FDs to efficiently reduce the search space. If the check becomes inefficient, the algorithm can return to the first stage and continue with all results discovered so far. This alternating two-phase detection strategy clearly outperforms all existing algorithms in terms of run time and scalability, while ensuring the discovery of all minimal FDs.

In [19], Z. Wei and S. Link noted that in the HyFD algorithm, strategy switching occurs when the current strategy performs incorrectly, but this switching occurs without evidence that the other strategy will perform well. Moreover, none of the previous algorithms have a mechanism for balancing the efficiency of execution time and main memory usage. The authors of [19] proposed a new hybrid algorithm DHyFD, in which the switch from the attribute-based approach to the row-based approach occurs whenever there is a probability of checking many FDs. This probability is estimated using the proposed measure. Setting lower thresholds for this measure allows the algorithm to use more main memory resources whenever this can improve the execution efficiency. Z. Wei and S. Link consider another drawback of the known algorithms for searching FDs between attributes in tables of Big Data systems to be an excessively large number of obtained FDs and a large total number of attributes in their left parts. Indeed, none of the known algorithms sets the task of minimizing the number of obtained FDs and the number of attributes in them. In the DHyFD algorithm, the method of constructing a canonical covering of the FD structure is used to reduce the number of obtained FDs, which actually constructs an elementary basis of the FD structure using the algorithm of C. Delobel and R.G. Casey [20]. In the work [21], A.A. Karpuk showed that, in the general case, this elementary basis of the FD structure is not minimal in terms of the number of FDs and is not optimal in terms of the total number of attributes in the FD, and proposed methods for constructing a minimal and optimal elementary basis of the FD structure.

In conclusion of our review of the main algorithms for finding FDs between attributes in Big Data system tables, we note the latest work by T. Bleifuss and T. Papenbrock et al. [22], in which they proposed a new hybrid algorithm FDhits. This algorithm integrates several optimization methods for efficient detection of FDs, namely a hybrid strategy, searching for unique attribute combinations through Hitting Set Enumeration, one-pass checking of all right-hand sides of FD candidates, and parallelization of computations. The authors [22] claim that the FDhits algorithm can find all minimal FDs in a table containing up to 100 attributes and more than 5 million rows in an acceptable time, for which all other algorithms cannot find a solution at all.

III. RESULTS

Based on the analysis of the main methods for extracting and algorithms for finding functional dependencies between attributes in Big Data tables, it can be concluded that research in this area is far from complete and should be continued. Clearly, there is no point in improving existing algorithms for finding functional dependencies between attributes in tables based on attribute lattice analysis or table row analysis, as their scope of application is significantly limited. Efforts should be focused on improving existing and developing new hybrid algorithms for finding functional dependencies between attributes in Big Data tables. First and foremost, these algorithms should incorporate methods for minimizing the number of functional dependencies obtained and the number of attributes within them, as well as more efficient methods for finding unique attribute combinations.

REFERENCES

[1]. Serra J. Deciphering Data Architectures. Choosing Between a Modern Data Warehouse, Data Fabric, Data Lakehouse, and Data Mesh. – O'Reilly, 2024. – 252 p.

[2]. Elmobark N. A Comprehensive Framework for Modern Data Cleaning Integrating Statistical and Machine Learning Approaches with Performance Analysis //AIandDataScienceJournal. – Vol. 1, No 1. – 2024. – P. 20–28.

[3]. Макаров А.В., Намиот Д.Е. Обзорметодовочисткиданныхдлямашинногообучения // InternationalJournalofOpenInformationTechnologies. – Vol. 11, No 10. – 2023. – P. 70–78.

[4]. Сравнение ведущих облачных платформ: AWS, GoogleCloud и MicrosoftAzure [Электронный ресурс]. – Режим доступа: https://playsdev.com/ru/blog/luchshie-oblachnye-reshenija-aws-google-cloud-i-azure – Дата доступа: 15.08.2025.

[5]. Armstrong W.W. Dependency Structure of Data Base Relationships // Proc. IFIP Congress. – Geneva, Switzerland, 1974. – P. 580–583.

[6]. Chen P.P. The Entity-Relationship Model-Toward a Unified View of Data // ACM Transactions on Database Systems. – March 1976. – Vol. 1, No 1. – P. 9–36.

[7]. Карпук А.А., Острейко В.М. Построение информационно-логической модели предметной области при проектировании базы данных // Вопросы радиоэлектроники. Сер. ОТ. – 1981. – Вып. 12. – С. 23–28.

[8]. Karpuk A. Bringing the Subject Domain Ontology to Optimal Canonical Form // Open Semantic Technologies for Intelligent Systems: Research Papers Collection. Issue 8 – Minsk: BSUIR, 2024. – P. 237–242.

[9]. Discover Dependencies from Data - A Review / Liu J. [et al.] // IEEE Transactions on Knowledge and Data Engineering (TKDE). – Vol. 24, No 2. – 2012. – P. 251–264.

[10]. TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies / Huhtala Y. [et al.] // Computer Journal. – Vol. 42, No 2. – 1999. – P. 100–111.

[11]. Novelli N., Cicchetti R. FUN: An Efficient Algorithm for Mining Functional and Embedded Dependencies // Proceedings of 8th International Conference Database Theory, ser. ICDT. – 2001. – P. 189–203.

[12]. Yao H., Hamilton H.J., Butz C.J. FD Mine: Discovering Functional Dependencies in a Database Using Equivalences // Proceedings of 2002 IEEE International Conference on Data Mining, ser. ICDM. – 2002. – P. 729–732.

[13]. Abedjan Z., Schulze P., Naumann F. DFD: E_Cient Functional Dependency Discovery // Proceedings of the International Conference on Information and Knowledge Management (CIKM). – 2014. – P. 949–958.

[14]. Flach A., Savnik I. Database Dependency Discovery: A Machine Learning Approach // AI Communications. – 1999. – Vol. 12., No 3. – P. 139–160.

[15]. Lopes S., Petit J.M., Lakhal L. Efficient Discovery of Functional Dependencies and Armstrong Relations // Proceedings of the 7th International Conference on Extending Database Technology, ser. EDB. – 2000. – P. 350–361.

[16]. Wyss C., Giannella C., Robertson E. FastFDs: A Heuristicdriven, Depth-First Algorithm for Mining Functional Dependencies from Relation Instances // Proceedings of International Conference on Data Warehousing and Knowledge Discovery. – 2001. – P. 101–110.

[17]. Efficient Discovery of Functional Dependencies on Massive Data / Wan X. [et al.] // IEEE Transactions on Knowledge and Data Engineering. – Vol. 36, No 1. – 2024. – P. 107–121.

[18]. Papenbrock T., Naumann F. A Hybrid Approach to Functional Dependency Discovery // Proceedings of the International Conference on Management of Data, ser. SIGMOD'16. – 2016. – P. 821–833.

[19]. Wei Z., Link S. Towards the Efficient Discovery of Meaningful Functional Dependencies // Information Systems. – Vol. 116. – 2023. – P. 102–224.

[20]. Delobel C., Casey R.G. Decomposition of a Data Base and the Theory of Boolean Switching Functions // IBM J. Res. AndDev. – 1973. – Vol. 17, No 5. – P. 374–386.

[21]. Карпук А.А. Анализ структуры функциональных зависимостей между атрибутами реляционной базы данных // Экономика и менеджмент систем управления. – 2017. – № 3 (25). – С. 64–70.

[22]. DiscoveringFunctionalDependencies throughHittingSetEnumeration / Bleifuss T. [et al.] // ProceedingsoftheACMonManagementofData (SIGMOD). – Vol. 2, No 1. – 2024. – Article 43, P. 43:1–43:24.