ISSN 2664-9969

UDC 004.042:004.623
DOI: 10.15587/2706-5448.2024.306435

Yehor Kucherenko,
Inessa Kulakovska

ANALYSIS OF METHODS AND ALGORITHMS FOR PROCESSING UNSTRUCTURED TEXT DATA BASED ON JSON TECHNOLOGY

The object of research is the process of automating systems for structuring data from several sources. The subject of the research is methods and algorithms for implementing a complete system for automated and parallel processing, validation and structuring of data. One of the most problematic areas is the merging of databases with different structures and several common fields into a generalized structure. The research was aimed at developing a system to increase the efficiency of automation of big data processing.

As a result of the work, optimization methods were studied, the influence of their internal parameters on the operation of algorithms was analyzed, their main advantages and disadvantages were determined, and software was developed in which the corresponding methods were implemented. An algorithm for structuring data before processing has been obtained. Data structuring is achieved by performing the «mapping» operation. Mapping can take place by indexes of already cleaned data or using a defined dictionary with a given set of keys, which allows not to care about the sequence of storing values and their possible shift.

The practical significance of the developed system lies in the improvement of methods of collecting and processing information for the purpose of its further validation, cleaning and accumulation in the following categories: geographic addresses and geo-coordinates, validation and automated addition of a mobile phone number to the international format, processing of car numbers (in modern and outdated format), VIN code of the engine and car brand, validation of urls of social networks, passport data and processing of personal data. Compared to similar methods for processing large volumes of data, the possibility of splitting the input file or stream into separate parts was used, the cleaned data from which is combined at the end of the system operation. Thanks to this, it is possible to process data whose size exceeds the available volume of the device's RAM, and the method of working with loosely structured text files in CSV format has been improved.

Keywords: *validation, intelligent system, unstructured data, JSON, CSV, crowding, ETL, ELT, automated system.*

Received date: 25.04.2024
Accepted date: 18.06.2024
Published date: 21.06.2024

How to cite
Kucherenko, Ye., Kulakovska, I. (2024). Analysis of methods and algorithms for processing unstructured text data based on JSON technology. Technology Audit and Production Reserves, 3 (2 (77)), 10–18. doi: https://doi.org/10.15587/2706-5448.2024.306435

1. Introduction

In order to process huge amounts of data, Facebook has developed a proprietary method known as Presto. There are various organized, unstructured, and different data file formats available, such as CSV, Image File Format, plain text, binary, XML, JSON, HTML, Excel, PDF, and others, from which it is possible to extract data based on requirements. Data management and organization is more important than ever before, because today the whole world is data-driven and only those organizations that have realized the huge impact of data analytics in time dominate and lead the information industry [1].

For the purpose of storing data and information, XML is used as a standard format in several organizations [2]. There are various reasons behind XML as a data exchange format on the Internet. For example, XHTML has been defined as a formal XML format that is successfully parsed by most HTML parsers [3].

JSON (JavaScript Object Notation) is a lightweight data exchange format. It is easy to read and write data that is generated and analyzed by computers. It is a language-independent text script that works on the syntax of the C language family [4].

The aim of this paper is to study the basics and features of creating an autonomous system for processing large amounts of data. In order to implement the software, it is necessary to analyze arrays of information from different subject domains and identify their features for the subsequent creation of rules for validation and processing of input values. The following steps are required for implementation:

– analyze methods and systems for automated processing of large data sets;
– develop principles of automatic partitioning and processing of text files whose size exceeds the available RAM capacity;

– provide a definition and justification of the basic principles and methods of working with poorly structured text files in the CSV (Comma Separated Value) format.

Data reading and initial preparation is a critical phase in data processing. It's here that the data structure is established, subject to transformation. Attributes like character encoding and separators are defined during this stage. Structural and technical constraints, such as limits on the available ROM space, are also defined at this stage. When reading data, an important indicator is the format of the input file – for example, a simple text file or a SQL dump, depending on which it is necessary to use different processing techniques, which further affects the speed of the system.

2. Materials and Methods

The object of research is the process of automating systems for structuring data from multiple sources. *The subject of research* is the methods and algorithms for implementing an integrated system for automated and parallel processing, validation and structuring of data.

The research was aimed at developing a system to improve the efficiency of automation of big data processing. As a result of the work, the optimization methods were investigated, the impact of their internal parameters on the operation of the algorithms was analyzed, their main advantages and disadvantages were identified, and software was developed in which the relevant methods were implemented.

To achieve the set goal, the following tasks were solved. The analysis of existing methods, models and algorithms for the implementation of a complete system for automated and parallel processing, validation and structuring of data, taking into account various data formats and possible associated uncertainties, to solve the tasks of organizing large data, was performed. The method of synthesis of Json processing information technologies was used to solve the problems of data analysis and modeling in the presence of various formats of data sources. A method of data filtering, their normalization and standardization has been developed based on a systematic combination of data transformation methods, a data supplement processing method has been implemented that identifies gaps in data, reveals patterns of their occurrence and forms data sets without gaps.

A laptop with an Intel Core i5 processor and 8 GB of RAM was used for the experiments. This equipment provided sufficient performance to process large amounts of data, although it had its limitations. The software consisted of Python 3 and virtualenv, which allowed to create isolated environments for executing scripts and ensured stability and controllability of program code execution. Using virtualenv was critical to avoid conflicts between libraries and dependencies.

Designed and tested tools for solving applied tasks to increase the efficiency of automation of big data processing and decision-making in various fields, confirming the reliability of scientific and practical results.

The main theoretical method was the use of hashing to filter data, which allowed for efficient processing of large volumes of information. Hashing was used to speed up searches and reduce the amount of memory needed to store data. The experiments were conducted under conditions of limited RAM, which was smaller than the input volume of data, which required optimization of the use of resources and efficient memory management. This created certain challenges, but also made it possible to test the effectiveness of the proposed methods in real conditions. This approach allowed to obtain results that show how it is possible to work with large data sets on limited hardware resources, which is important for the further development of data processing methods.

3. Results and Discussion

3.1. Using programming technologies in data mapping. It is convenient to represent the abstraction of input data mapping using the concepts of OOP (Object Oriented Programming) and classes. To simplify the work with data, it is possible to create a «Unit» class.

An ETL (Extract, Transform, Load) unit is an object that has the following properties:

– unit service name – a string field – used at the stage of debugging and logging. For example, a unit created for processing and validating mobile phone numbers can have the service name «Phone»;
– data conversion category – phone numbers, full names, links to web resources – data and other categories have different methods and algorithms for processing raw input data. The category name specified when creating a unit object can be used as a key to the dictionary that will be specified at the stage of input data mapping;
– a subcategory in this category;
– a list of called functions for transforming input data – in a given sequence of list items;
– a dictionary with data to clean the string before conversion.

3.1.1. Features of the use of software architecture. Accounting for key fields. In some cases, when operating an ETL system, it is important to take into account such a factor as key fields. The peculiarity of key fields (columns) within a single row is that there is a predefined condition (combinatorial) under which the row that contains either a certain number of key fields or a combination of them is considered valid after cleansing. There are two approaches to solving this problem [5–7].

1. Define a list of dictionary keywords in advance and then make combinations of a certain length from them. This method has a significant drawback: if the final list of dictionaries is large and/or the number of key fields is large enough to produce long chains of combinations, the full search takes a long time [8, 9].

2. Keep track of key fields at the processing stage – using the key field counter for each row. During the processing of each new row, the key field counter resets to zero. Then, each column is handled following ETL rules [10, 11]. If the processed value aligns with business logic and the ETL unit marks it as a key, the key field counter increments. If the key field counter is greater than one after processing all columns in a row, the row is considered valid. This approach allows to add to the final set of rows only valid rows that have exactly the key fields, which avoids situations when the final row after processing contains non-empty columns, but they are not key fields, and the row is useless.

3.1.2. Flexibility of working with ETL units. The basis for working with ETL units is a predefined dictionary that contains ETL unit objects as values. The advantages of precreated objects are as follows. Unambiguity – it is assumed

ISSN 2664-9969

that the objects themselves will not change, but it is possible to expand the list of methods used by this object [7, 11].

Consider the parameters of its constructor.

The first parameter is the service name of the unit. It is used for two purposes:

1. As service information during logging and debugging.
2. As a key for the dictionary of cleared columns.

The second parameter is the value of the enumeration class – data categories. It is the use of the structure of nested dictionaries (subcategories in a category) with lists of functions attached to each subcategory that allows to maintain the flexibility of using an ETL unit by expanding or reducing the list of functions to be called, and the functions themselves can (and should) be located in separate external files and imported into the file in which this dictionary is defined, which allows to radically change their behaviour without changing the structure of the dictionary and the ETL units themselves.

The third parameter is the name of a subcategory in a given category. A subcategory is a dictionary key in the dictionary of a given category, which is already used to retrieve a list of functions for cleaning. As a rule, the service name of the unit and the name of the subcategory coincide for logical reasons.

The fourth parameter is the character dictionary that is used when cleaning the input data for this unit.

3.1.3. The generalized structure of the units. The main idea is as follows:

1. A unit is created once in an external file and remains unchanged. To work with a unit, it is imported and called by the specified key in the unit dictionary.
2. The unit dictionary remains unchanged and is located in an external file, which ensures the readability of the rest of the code.
3. When creating a unit, it is based on the concepts of category and subcategory.
4. All categories and their subcategories are rigidly defined and predefined in the form of a hierarchical structure of dictionaries – in an external file.
5. The only changeable element of ETL units is the list of functions that are called for a given subcategory. This allows to change only the number of items in the list for a given subcategory – without affecting the logic of the unit class or the initial dictionary of the created unit objects.

The general structure of calls: call an etl unit by key -> access the constant dictionary of pre-created units -> access the constant hierarchical dictionary of categories and their subcategories -> get a list of functions to be called for a given category and subcategory (and, accordingly, when using this unit), Fig. 1.

Advantages of a unit class over other data structures. The advantage of using a unit object over using a regular dictionary should be noted right away:

– the use of a dictionary does not allow to keep track of key fields – without additional auxiliary data structures (lists or other dictionaries);
– the use of a dictionary does not allow to immediately specify a dictionary of unwanted characters to be used for this column (without additional auxiliary data structures (lists or other dictionaries));
– the use of a dictionary – without a two-level binding structure (a dictionary of units and a dictionary of categories) – does not allow for a clear division of data into categories, which complicates the work with the code when working with a large number of columns (entities).

3.1.4. The ability to disable the unit. In the unit class – there is a field – «unit_is_disabled» – set to «False» by default: «self.unit_is_disabled=False». This field allows to «disable» a unit before using it, before binding it to a column, which in turn allows to do the following actions:

1. Keep the specified column unchanged, i. e., do not process it – if necessary.
2. Continue to use the service name of the unit as a key for the dictionary of the cleared columns – during data processing.

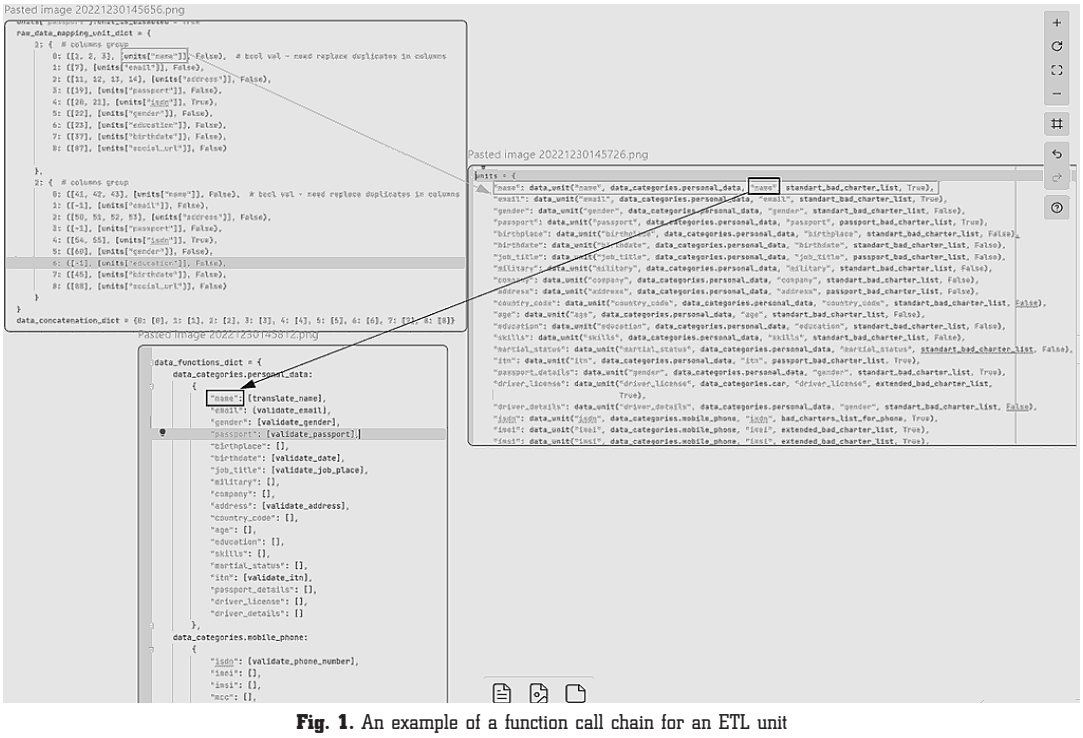

Fig. 1. An example of a function call chain for an ETL unit

Since all unit objects are pre-created and stored in the dictionary, to «disable» a unit, it is necessary to access the dictionary using the required key and set the value of the «unit_is_disabled» field – for the unit – to «True».

3.1.5. Partially disable a unit. This option of disabling a unit allows to leave the column value unchanged – except for removing characters from the list of bad characters.

It is also important to note that if there are a large number of columns in the input file (and, accordingly, units to clean), it is inconvenient to manually set the partial disable option for each unit. That is why it is advisable to provide a partial disable flag for all units, which, if set, will set the value of this parameter for each unit in the body of the ETL system object constructor. This will allow to disable all units and enable only the necessary ones, thereby inverting the logic [12].

3.1.6. Parameterization of the unit. The following function has been added for the unit. This function accepts as a parameter – a dictionary, where key – the name of the parameter to be set and value – the value of this parameter. If the keys in the passed dictionary are from the sand of allowed parameters for this unit, the values from the dictionary passed to the function are set as the values of the unit's settings dictionary, which are then, in turn, used in the data verification and processing function.

When a unit is created, it's assigned a predefined dictionary of allowed parameters called «unit_params». These parameters are then checked in the «set_unit_params» method, where if a key is found, its corresponding parameter is set to the specified value. These parameters and values are subsequently utilized in validation functions, exemplified by the date format validation function shown in Fig. 2.

In the example, it is possible to see that in the original data, one of the columns contains two types of information that can be represented in the system as separate columns – «passport» and «passport_details». In this case, a tuple of indices is specified as the index (dictionary key). The number of elements in the tuple must be the same as the number of units in the list to be processed, which are sequentially in the order in which the data in the original column is displayed, separated by an additional delimiter. In this example, the original column contains passport information followed by additional details. The column is split using the additional separator, with each part processed by a corresponding unit and added to the corresponding index in the index tuple as a column in the cleaned file. That is, the part of the original column that contains the information «passport» will be added as a column in the cleaned file, and the part that contains the information «passport_details» will be added.

3.2. Deleting duplicates. Types of value checks to remove duplicates:

– horizontal check – involves comparing columns – within the same row – for duplicate values;
– vertical check – a row-by-row check of values – for a specific column. If the value in a column or columns matches in two rows, the rows are considered duplicates. If there are several columns to check, if at least one of them matches in two rows, it is considered a duplicate;
– combining vertical and horizontal checking – means vertical checking, provided that the value matches for all the columns listed for duplicate search. If there are several columns to check (Fig. 3), a row is considered a duplicate only if all the specified columns in different rows (vertical check) match each other (horizontal check).

3.2.1. Types of duplicate deletion. Duplicate deletion can be divided into the following categories:

– deleting complete duplicate rows – at the data reading stage;
– deleting duplicates by columns – at the processing stage;
– primary duplicate removal with combining the values of row columns by the key field to be selected (includes sorting the row and searching downwards for duplicate «co-joins» with combining the values with them and marking them as duplicates).

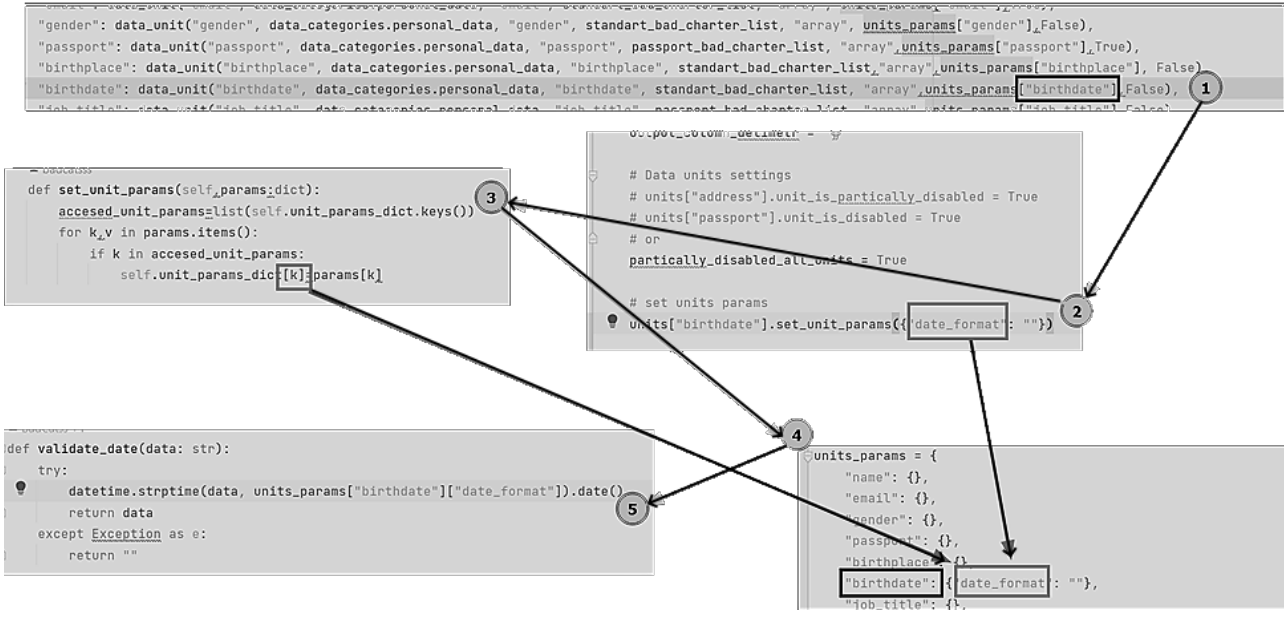

Fig. 2. An example of a unit parameterization sequence

ISSN 2664-9969

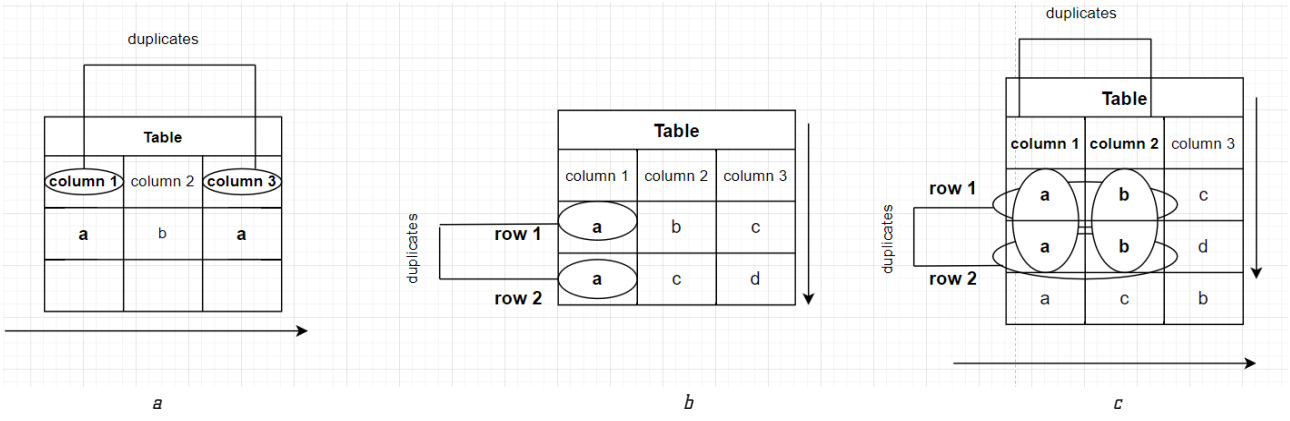

Fig 3. Types of duplicate data: *a* – horizontal duplicates check; *b* – vertical duplicates check; *c* – combining vertical and horizontal checking

Duplicate removal using a specified key field without merging columns. This process involves a dictionary with category keys such as «name», «email», and «isdn». Each category key contains sub-dictionaries where the key is the field value (e. g., phone number) and the value is the entire dictionary object for that category.

This approach allows to perform a quick search in the LARGEST category of keys – the global category, without sorting and comparing only two neighboring objects.

Category dictionary (name, isd, email) -> category-specific dictionary -> key – value belonging to this category: value – full dictionary object containing this key.

Checking for duplicates by hashing – before writing to a file (checking the hash of a string in a file with hashes of previously written strings).

3.2.2. Merge «duplicate» rows by key field (Basic removal of duplicates with the merge of column values of rows by the selected key field). The data after cleaning can be filtered in the following ways:

1. Simple deletion of completely duplicate rows – for this purpose, it is enough to use the standard data structure – «set».

2. Combining two rows – «column to column» – if a common key column is found (a column in which the values for these rows are the same). It is worth noting that the choice of such a column should be made accordingly – it can be, for example, an email address or a passport number – that is, columns that, by definition, are unique within the same data set. Let's look at the second case.

When to merge data by a key column, it is possible to face the problem that such rows can be quite far apart, which leads to the problem of comparing each row with each lower row. To solve this problem and optimize the work, it is necessary to sort the rows by the desired key column, thereby placing the rows with the same columns side by side, which will allow to check the similarity of a row with only one row below it (i. e. N+1). If to find a row that duplicates a key field, the data of the columns of this row are merged with the data of the corresponding columns of this row, and the lower row that was a duplicate is removed from the list of rows.

Not all columns require concatenation for duplicate detection. For instance, if «email» serves as the unique key, we won't typically concatenate «name» values. However, phone numbers might need concatenation in the final result row. To solve this problem, it is necessary to define a list of fields to be skipped in advance.

When gluing columns into one, the following options are also possible:

1. Simple joining of columns through a separator.

2. Splitting each column by the specified separator, converting them into sets, combining the two sets, and finally gluing the elements of the resulting set – through the separator – into the final row – the column value.

The first option is ideal for simple values like residential addresses. These values can be compared as integer strings rather than sets, preserving their order. Even using «frozenset» won't retain meaning when merging two sets with deleted duplicates and new elements added.

«21, B. Morskaya St.» and «21, Bolshaya Morskaya St.» – which will result in the line: «street B. Bolshaya Morskaya str. house 21»)), and the second option is suitable for filtering values in a column, for example, when combining columns with mobile phone numbers.

When joining rows by columns, ensure columns aren't empty, meaning they're not equal to « » or «\n». Allowing this could result in merging rows due to the structure of cleared data, where dictionaries in each row under the specified key for duplicate clearance might contain empty strings or newline characters.

3.2.3. Types of missing values – when merging duplicates. When merging column values with duplicate rows, the columns can be divided into two categories:

– columns – for which duplicate rows are checked;

– columns whose values will be merged if the values of the columns of the first type match.

There are the following options for skipping column merging when searching for duplicates:

1. Skip columns – for merging. The list of columns is used – «merge_duplicates_skip_merge_columns». The values of the columns specified in this list are not merged through the delimiter in case of duplicates by columns used to check for duplicates. The value of the first row of two is used, the column value of the second row is discarded. When using this type of skip, the values of duplicate rows are not merged at all. This type of gap ignores the column to be merged.

2. Column skip – for merging – by column value. When using this type of skip – column – is not ignored for merging – the combination of column values – for duplicate rows – is performed if the column value itself is not blacklisted. *This type of skip – ignores the column value for merging.*

Consider the extreme case when the key field for merging duplicates is empty (equal to «""») or equal to a newline character, or the value of the column to be merged (but not

the column itself) is blacklisted (see more details in «types of column merging»).

For this case, an additional «else» branch was added – which, if triggered, loops through an additional list of fields that can be used to check rows for duplicates. At the same time, since the key field for checking changes with each iteration, it is also necessary to return the value of the main field to it.

The logic of the «replace_duplicate_value» method is given below.

1. Two additional data structures are created.

2. «The dictionary «founded_duplicates_for_uniq_fields» stores sets of identified duplicate values. If triggered, «founded_duplicates_for_uniq_fields_indexes» contains pairs: the unique field value added to «founded_duplicates_for_uniq_fields» and the row index where this value occurs».

It is worth explaining separately why the values are replaced with a stub instead of deleting the entire dictionary object from the list of strings (dictionary list). The fact is that when we are inside a loop and try to delete a list item, we violate the index order, which further leads to an out-of-array result.

The following options are possible when combining strings:

– the strings are equal – no concatenation is needed – it is possible to return any of the strings (the condition «if value_by_key!=second_value_by_key» is NOT met);

– the strings are not equal and both strings are valid (the condition «if value_by_key!=second_value_by_key:» is met);

– one string is a subset of the second – return the second string (the length of the symmetric set is zero: Condition is NOT met: «if len(string_diff)>0»);

– one string is NOT a subset of the other – merge the strings (the length of the symmetric set is NOT zero: the values of the string components do not overlap completely to consider one of the strings a complete substring of the other);

– the strings are not equal, but one of the strings is not valid (in our case, one of the strings is equal to the stub value – «FOR REMOVE») – return only the valid string (the condition is NOT met – «if value_by_key!="FOR REMOVE" and _value_by_key!="FOR REMOVE"»).

Important: the «need_filter_then_merge» parameter specifies whether to return a difference merge (merging of rows, excluding intersections) or just return the merged rows.

3.2.4. Removing duplicates before writing to a file (duplicate search post-processing). Secondary removal of duplicates by the selected key field – without combining column values:

– checking all sorted values by the specified key field («remove_duplicates_by_column»);

– checking all sorted values by a separate list of fields – to remove duplicates before writing to a file («post-processing_duplicates_remove_keys»).

The main difference between the updated postprocessing duplicate search and the main duplicate search is that there is no sorting of the list of values by the specified column. The entire search for duplicates at the post-processing stage consists of comparing only two rows – i and i+1.

Removing duplicates by hashing – before writing to a file.

Duplicate search can also be divided into:

– local duplicate search – within a given file;

– searching for duplicates during cleaning;

– search for duplicates at the post-processing stage;

– searching for duplicates in relation to an external database. Example diagram at Fig. 4.

«When searching for duplicates in an external database, the challenge lies in its black box nature; we lack information on loaded rows without access to loading logs».

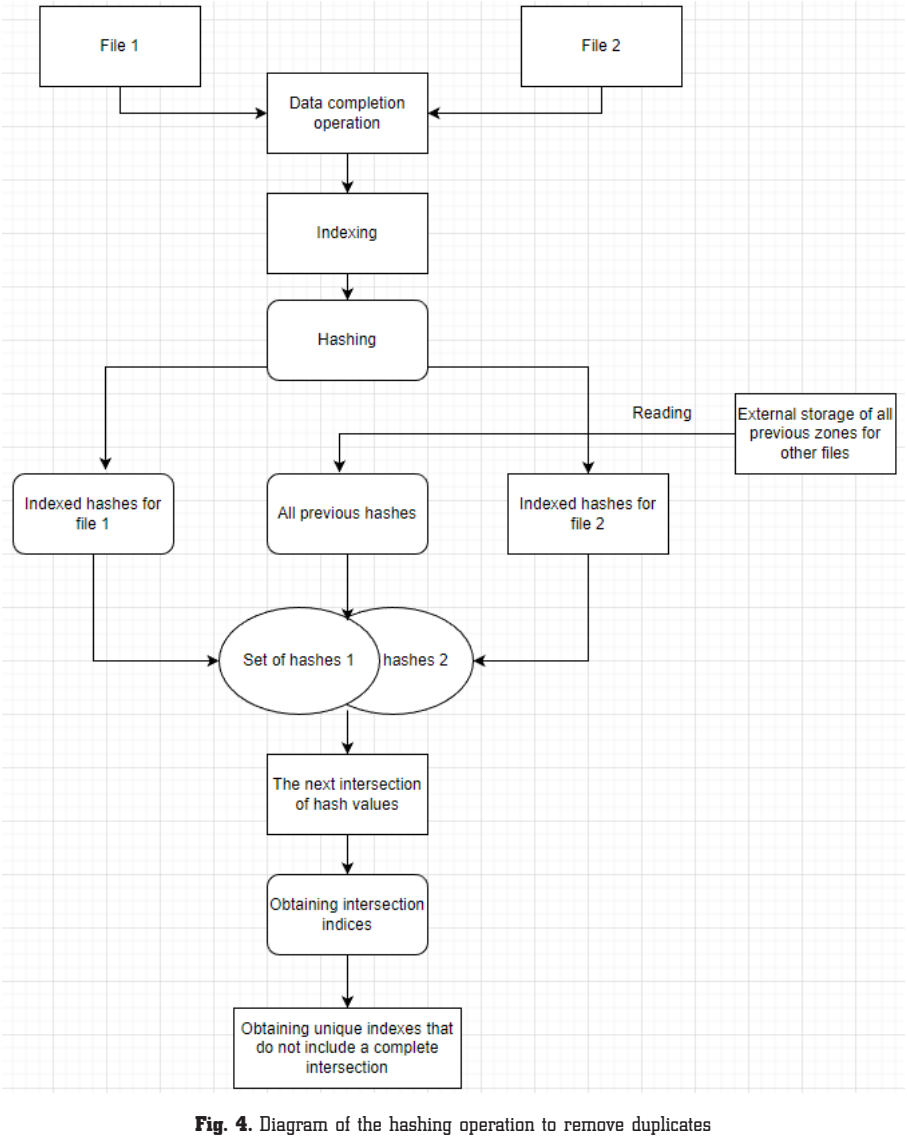

Fig. 4. Diagram of the hashing operation to remove duplicates

ISSN 2664-9969

Let's look at the problem in more detail. The task is to search for non-overlapping values in two lists of dictionary objects. Each list can contain at least 100 thousand objects, and this is only during the current operation of loading and processing new values, and all previously loaded objects must also be taken into account.

This task can have two initial starting conditions:

1. More than one file is input.
2. The input is a single file.

Let's consider each of the cases. More than one file is given as input. In this case, the algorithm consists of the following steps:

1. Reading a pair of files (more about reading files below and example diagram at Fig. 5).
2. If the lengths of the lists of objects for the two files are not equal, perform the operation of supplementing the smaller list with empty dictionaries.
3. Perform indexing – create a pair – the index of the object in the list and the object itself.
4. Hash the dictionaries by their values.
5. If there is a file with previous hashes, honor it.
6. Take the non-intersection of the sets of all hashes.
7. Find the object indices for the remaining hashes that do not intersect (reverse indexing).
8. Get unique dictionaries based on the found indices.
9. Update the file with all hashes – writing the hashes of all dictionary objects – from the read files.

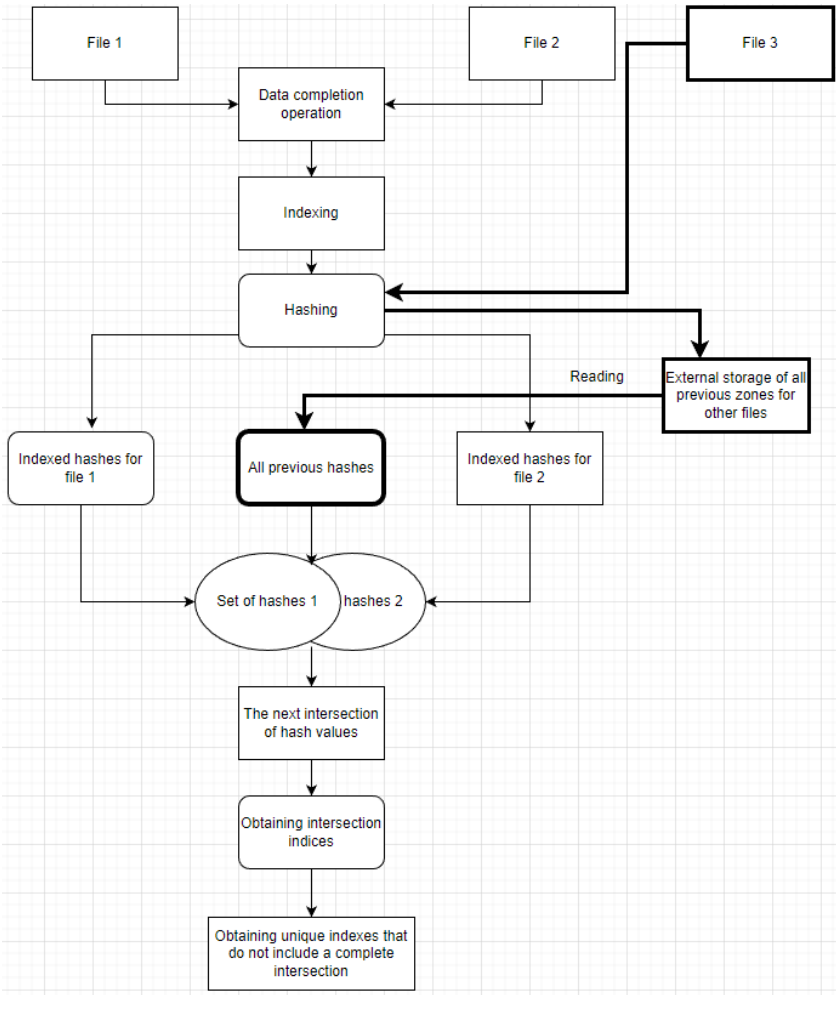

Fig. 5. Diagram of the hashing operation to remove duplicates if there are more than two input files

For the case when there is only one input file:

1. Add indexing and hashing of all dictionaries.
2. Write the resulting list of hashes to a file.
3. Return the list of all objects.

When reading a list of files in a loop, it is possible to get a sliding window. When reading in pairs, there may be a problem of re-reading already read data when the window is moved. To avoid this problem, it is necessary to memorize the paths to the already read files, and if the pair that got into the sliding window contains an already read file, then read only one unread file, thereby reducing the task to the second option-searching for unique (Fig. 6).

3.3. Mapping of cleansed data. ETL system – can map data in two «modes» of operation:

1. Data mapping with the final output of a list of columns, which will then be glued into a row.
2. Data mapping with the final output of a column dictionary, which will then be glued into a string.

Each option has its advantages and disadvantages:

– The option of using a list can lead to a shift in the list items, in case of incorrect manipulation of the item – for example, one of the items in the final list became an empty string, which is why it was discarded at the final stage of verification before concatenation. One issue arises from the variable size of the list, where splitting and merging can distort the count. Using a dictionary eliminates shifting problems but requires predefining keys and values, complicating the process.

As it is possible to see, the option of using a dictionary is preferable. It is responsible for automatically extending the dictionary to the required length if the number of initial cleansed columns is greater than the number of specified concatenation dictionary items.

The updated version of the function takes into account the columns that were obtained as a result of the expansion operation – the «full_keys_list_for_reamer_adding» parameter. The columns obtained as a result of the sweep operation are added to the temporary dictionary with empty values, after which the temporary dictionary is added – based on the passed dictionary with cleared values.

Specialized functions – functions for data verification and validation, for checking data from a certain category, for example, the function for checking a mobile phone number, the function for checking the format and boundary values of the date of birth, the function for checking the address.

1. Meta-functions – a type of function that does not belong to specialized functions. Generalized functions whose logic does not depend on the data category. An example of such a function is the «remove_column_by_value» function, which replaces the entire column in a csv file if the text in the column starts with a certain substring.

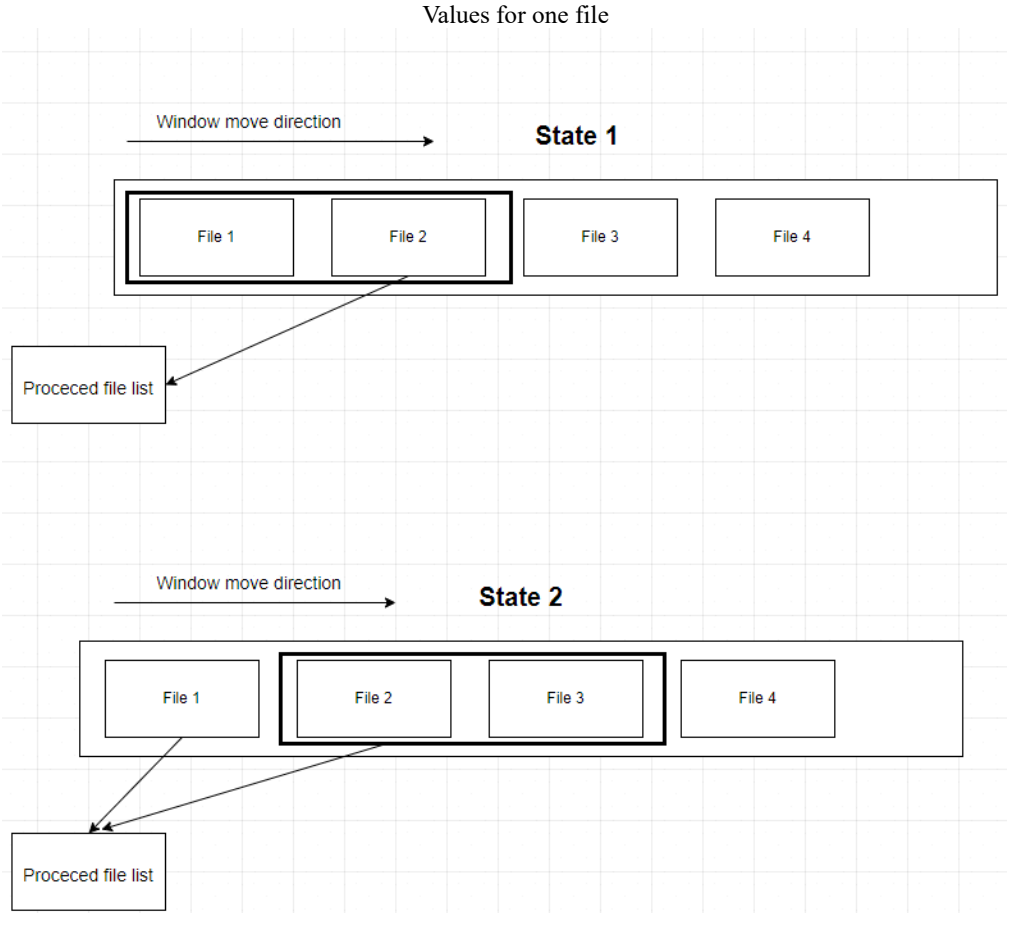

Fig. 6. Using the sliding window approach when performing a duplicate search operation using hashing

2. Dictionary search functions are one of the main functions that are used in the same way as auxiliary functions for specialized functions. The basic idea of these functions is as follows: set an initial dictionary of words to search, for example:

– all values in this dictionary are specified in a single case, for example, lowercase. For this dictionary, another auxiliary dictionary is created;

– it contains the initial number of keywords in each list for a given group. As it is possible to see, the keys of these dictionaries coincide, and the second dictionary contains the number of items in the list of the first dictionary for the same key.

Then, in the process of specialized functions, they can use the «*keywords_for_search*» dictionary to search for occurrences, start or end of their data in the list – by the key of this dictionary. The main idea of the dictionary is to further supplement it with all possible combinations, for example, using such a function:

Furthermore, to conserve resources, this function is invoked only when the number of items in the list matches the value in the «start_keywords_dict_length» dictionary for the corresponding key, indicating the list hasn't been fully augmented with combinations yet. This is exactly why the «*start_keywords_dict_length*» dictionary was created – in fact, its values serve as a check whether it is necessary to call the function of adding a list in the dictionary. «After updating the dictionary, the search functions verify the specified value under the given key in the dictionary».

3. Validation functions are a subset of dictionary search functions and, in fact, are completely based on their use. The only difference that can be distinguished for such functions is the ability to replace the found values (occurrences) after a dictionary search with another specified value.

4. Transformation functions are essentially the same as validation functions, but they are placed in a separate group because, unlike the first ones, it is possible to apply more complex (any) transformations to the data, not just a simple replacement of the found occurrences.

An example of using hierarchical functions:

```
'''python
def validate_passport(data: str):
return check_substring_in_data(data, "passport", True)
'''
```

«validate_passport» – a specialized function; «check_substring_in_data» is a validation function that utilizes the dictionary search function (enhancing the dictionary and retrieving its value by key, a comprehensive list of keywords for search, through the «check_search_dict» function). During operation, it searches for one of the word variations, such as «passport», in the input data. The parameter «True» signifies the necessity to replace this occurrence at the beginning of the line with an empty line, effectively deleting it.

3.4. Discussion. Compared to similar methods, for processing large volumes of data, the possibility of dividing the input file or stream into separate parts was used, which allowed faster cleaning of the combined data. This allows to process data, the volume of which exceeds the available

ISSN 2664-9969

RAM of the device, and improves the methodology of working with unstructured text files in CSV format.

The practical significance of the developed intelligent system consists in improving the methods of collecting and processing information, its further verification, cleaning and accumulation in the appropriate categories. Data structuring is achieved by performing a mapping operation on the indexes of already cleaned data or using a predefined dictionary with a given set of keys, which allows not to worry about the sequence of storing values and their possible shift.

One of the most problematic areas is the unification of databases with different structures and some common fields into a common structure, and its incomplete automation. The semi-automated process of the system for structuring data from several sources is considered. At the beginning of the software, it is necessary to create appropriate dictionaries and rules for combining data. The resulting application architecture allows to quickly configure the system. Adding new functionality to the application takes place without making changes to the logic of the main data processing pipeline – by creating independent function handlers. These functions, according to their separate logic, can be added to the main list of handler functions in any order.

During the first year of the war in Ukraine, people moved from the temporarily occupied territories to Mykolaiv Oblast, Kherson Oblast and further across Ukraine by various means: trains, buses, motor vehicles. The lists of those who went with whom, how many people, how to contact them were in fairly free forms, to bring them into a common database, the need for appropriate software was quite in demand.

Further research consists in the need to use the basics of lexical analysis and tokenization of data, the work partially considers cases that can be considered borderline – for certain data domains, for example – processing the name of a person, which is inside an arbitrary line and with a previously unknown location relative to other tokens of this line.

4. Conclusions

As it has been demonstrated, structuring data before processing it is one of the key stages that affects the correctness of the final result. Data structuring is achieved by performing a «mapping» operation. The mapping itself can be performed by the indices of already cleansed data or using a predefined dictionary with a given set of keys, which allows not to worry about the sequence of storing values and their possible shift.

To improve the data processing process, namely, to make the data pipeline more flexible, the application of OOP principles was demonstrated in order to create independent objects that can encapsulate the logic of data processing from a particular information domain, taking into account the specifics of the data.

The principle of working with data separators was also considered, and it was demonstrated that data can be divided by the main and additional separator, which affects the logic of their further processing.

Conflict of interest

The authors declare that they have no conflicts of interest in relation to this study, including financial, personal, authorship, or other, that could affect the study and its results presented in this article.

Financing

The study was conducted without financial support.

Data availability

The manuscript has no associated data.

Use of artificial intelligence

The authors confirm that they did not use artificial intelligence technologies in the creation of the presented work.

References

1. Chaturvedi, S., Kumar, P. (2018). Extraction and Conversion of Web JSON Data into Pandas Data Frame by storing it into Text File using Python. *International Journal for Research in Applied Science & Engineering Technology, 6 (XI)*.
2. *JSON Community*. Available at: https://json.com
3. *Wikimedia Foundation. Wikipedia*. Available at: https://en.wikipedia.org/wiki/Wikimedia_Foundation
4. Frozza, A. A., Mello, R. dos S., Costa, F. de S. da. (2018). An Approach for Schema Extraction of JSON and Extended JSON Document Collections. *2018 IEEE International Conference on Information Reuse and Integration (IRI)*. doi: https://doi.org/10.1109/iri.2018.00060
5. Avramenko, O. M. (2017). Intelektualna systema obrobky nestrukturovanykh tekstovykh danykh na osnovi tekhnolohii JSON. *Visnyk Natsionalnoho tekhnichnoho universytetu «KPI», 64,* 44–48.
6. Babenko, L. P. (2018). Zastosuvannia JSON dlia obrobky tekstovykh danykh v informatsiino-poshukovykh systemakh. *Naukovi zapysky Natsionalnoho universytetu «Lvivska politekhnika», 843,* 34–39.
7. Honcharuk, L. V. (2019). JSON: suchasnyi format obminu danymy. *Visnyk Kyivskoho natsionalnoho universytetu imeni Tarasa Shevchenka. Seriia: Kompiuterni nauky, 14,* 5–10.
8. Elsayed, K. I., Elgamel, M. S. (2020). Web of Things Interoperability Using JSON-LD. *2020 30th International Conference on Computer Theory and Applications (ICCTA)*. doi: https://doi.org/10.1109/iccta52020.2020.9477674
9. Sun, C., Zeng, X., Sun, C., Si, H., Li, Y. (2020). Research and Application of Data Exchange based on JSON. *2020 Asia-Pacific Conference on Image Processing, Electronics and Computers (IPEC)*, 349–355. doi: https://doi.org/10.1109/ipec49694.2020.9115155
10. Xu, J., Deng, H. (2021). JSON-ASR: A lightweight data storage and exchange format for automatic systematic reviews of TCM. *TMR Modern Herbal Medicine, 4 (2),* 12. doi: https://doi.org/10.53388/mhm2021a0401001
11. Afsari, K., Eastman, C. M., Castro-Lacouture, D. (2017). JavaScript Object Notation (JSON) data serialization for IFC schema in web-based BIM data exchange. *Automation in Construction, 77,* 24–51. doi: https://doi.org/10.1016/j.autcon.2017.01.011
12. Garg, I. (2024). *Study on JSON, its Uses and Applications in Engineering Organizations*. doi: https://doi.org/10.13140/RG.2.2.19850.07367

✉ ***Yehor Kucherenko,*** *Department of Intelligent Information Systems, Petro Mohyla Black Sea National University, Mykolaiv, Ukraine, e-mail: yehor.kucherenko@chmnu.edu.ua, ORCID: https://orcid.org/0009-0008-0909-3780*

Inessa Kulakovska, *PhD, Associate Professor, Department of Intelligent Information Systems, Petro Mohyla Black Sea National University, Mykolaiv, Ukraine, ORCID: https://orcid.org/0000-0002-8432-1850*

✉ *Corresponding author*