1 Introduction

This document will provide a walk-through of the consolidate script with dummy data, aimed to provide clarity and understanding to how the script works.

At a high-level, this script reads an input csv file and constructs a compressed csv (and json) representation for it. In the compressed csv, each cell can represent a set of values rather than a singular value.

2 Data

Assume the input data that we want to consolidate is as follows:

GuardiumVersion	DatabaseName	DatabaseVersion	OSName	OSVersion	Feature
11.0	Cassandra	Cassandra 3.11.10	Ubuntu	Ubuntu 21.04	A
11.0	Cassandra	Cassandra 3.11.10	CentOS	CentOS 7	A
11.0	Cassandra	Cassandra 3.11.10	CentOS	CentOS 8	A
11.0	Cassandra	Cassandra 3.11.10	CentOS	CentOS 8	A
11.0	Cassandra	Cassandra 3.11.10	Windows	Windows 10	A
11.0	Cassandra	Cassandra 4.0	Windows	Windows 10	B
11.1	DB2	DB2 11.5.7	SUSE	15	C
11.1	MariaDB	MariaDB 10.5	macOS	macOS 11.0	C
11.1	MariaDB	MariaDB 10.6	Ubuntu	Ubuntu 22.04	C
11.1	MongoDB	MongoDB 4.0	CentOS	CentOS 6	B
11.1	MongoDB	MongoDB 4.0	CentOS	CentOS 7	B
11.1	MongoDB	MongoDB 4.0	CentOS	CentOS 8	B
11.2	MongoDB	MongoDB 4.2	CentOS	CentOS 6	B
11.2	MongoDB	MongoDB 4.2	CentOS	CentOS 7	B
11.2	MongoDB	MongoDB 4.2	CentOS	CentOS 8	B
11.2	MongoDB	MongoDB 4.4	CentOS	CentOS 6	B
11.2	MongoDB	MongoDB 4.4	CentOS	CentOS 7	B
11.2	MongoDB	MongoDB 4.4	CentOS	CentOS 8	B
11.3	MongoDB	MongoDB 4.6	CentOS	CentOS 8	B
11.4	MongoDB	MongoDB 4.6	Windows	Windows 10	B
11.3	Redis	Redis 7.2	Debian	Debian 11	A

Table 1: Input Data From CSV

This script will compress that data to find this representation

GuardiumVersion	DatabaseName	DatabaseVersion	OSName	OSVersion	Feature
11.0	Cassandra	Cassandra 3.11.10	CentOS	CentOS 7, CentOS 8	A
11.0	Cassandra	Cassandra 3.11.10	Ubuntu	Ubuntu 21.04	A
11.0	Cassandra	Cassandra 3.11.10	Windows	Windows 10	A
11.0	Cassandra	Cassandra 4.0	Windows	Windows 10	B
11.1	DB2	DB2 11.5.7	SUSE	15	C
11.1	MariaDB	MariaDB 10.5	macOS	macOS 11.0	C
11.1	MariaDB	MariaDB 10.6	Ubuntu	Ubuntu 22.04	C
11.1	MongoDB	MongoDB 4.0	CentOS	CentOS 6, CentOS 7, CentOS 8	B
11.2	MongoDB	MongoDB 4.2, MongoDB 4.4	CentOS	CentOS 6, CentOS 7, CentOS 8	B
11.3	MongoDB	MongoDB 4.6	CentOS	CentOS 8	B
11.3	Redis	Redis 7.2	Debian	Debian 11	A
11.4	MongoDB	MongoDB 4.6	Windows	Windows 10	

Table 2: Compression version of Table 1

WalkThru Python Code

We will start our walk-through when we call the consolidate function in main() in runner.py

```
consolidate(output_json_path, output_csv_path, input_csv_path,
                    version_headers, full_key, feature_headers,
                    partition_header_number, logger)
# full_key = ["Guardium Version","DatabaseName","DatabaseVersion","OSName","OSVersion","Feature"]
# version_headers = ["Guardium Version","DatabaseName","DatabaseVersion","OSName","OSVersion"]
# feature_headers = ["Feature"]
# partition_header_number = 1
```

Here, the arguments *output_json_path, output_csv_path, input_csv_path* are just paths to where we're reading the input data and writing the compressed data.

The argument *full_key* is just all the header names. Whereas, *version_headers* represent those header names that represent columns whose values are related to the version of something. On the other hand, *feature_headers* are the header names related to a feature.

partition_header_number is the index of the column based on which you want to partition the data. Essentially, during the compression, you will split your input data rows based on the value in column at index *partition_header_number*. This will make more sense further on.

```python
# -------------------- Inside consolidate function --------------------
    # Instantiate output variables
    output_csv = []
    output_csv.append(full_key) # Add headers to first line of csv output
    output_json = {}

    # Get unique values in a column (eg.unique database names in database column)
    # (This will be used to partition the tabular data
    # before compression for efficiency)
    uniq_vals = read_csv_get_unique_vals_in_column(input_csv_path,partition_header_number)
    logger.debug("Stored all unique vals of column #:%s ", partition_header_number)

    # uniq_vals = ["MongoDB","Cassandra","DB2","MariaDB","Redis"]
```

uniq_vals is a set of all the unique values in the column at index = *partition_header_number*

```python
    # Loop thru each uniq value
    for uniq_val in uniq_vals:
        output_json[uniq_val] = [] # Will store consolidated data for specific uniq_val

        # Partition data (Get subsection of all rows with uniq_val in the partition_header_number)
        partitioned_data = read_csv_for_uniq_val(input_csv_path,uniq_val,partition_header_number)
        partitioned_data = remove_duplicates_2d(partitioned_data)
        logger.debug("Partitioned Data for %s",uniq_val)

        # uniq_val = "MongoDB"
        # partitioned_data =
        # [['11.1', 'MongoDB', 'MongoDB 4.0', 'CentOS', 'CentOS 6', 'B'], ['11.1', 'MongoDB', 'MongoDB 4.0', 'CentOS', 'CentOS 7', 'B'], ['11.1', 'MongoDB', 'MongoDB 4.0', 'CentOS', 'CentOS 8', 'B'], ['11.2', 'MongoDB', 'MongoDB 4.2', 'CentOS', 'CentOS 6', 'B'], ['11.2', 'MongoDB', 'MongoDB 4.2', 'CentOS', 'CentOS 7', 'B'], ['11.2', 'MongoDB', 'MongoDB 4.2', 'CentOS', 'CentOS 8', 'B'], ['11.2', 'MongoDB', 'MongoDB 4.4', 'CentOS', 'CentOS 6', 'B'], ['11.2', 'MongoDB', 'MongoDB 4.4', 'CentOS', 'CentOS 7', 'B'], ['11.2', 'MongoDB', 'MongoDB 4.4', 'CentOS', 'CentOS 8', 'B'], ['11.3', 'MongoDB', 'MongoDB 4.6', 'CentOS', 'CentOS 8', 'B'], ['11.4', 'MongoDB', 'MongoDB 4.6', 'Windows', 'Windows 10', 'B']]
```

We now use each *uniq_val* to get *partitioned_data*, which is the subsection of all rows with *uniq_val* in the *partition_header_number* column index. We continue in this for-loop with *uniq_val*="MongoDB"

```python
        # Group rows of data based on their feature values,
        # Returns Dict[str, List[List[str]]]
        # A dict with a string key (representing one rows' feature values) and,
        # Lists of lists of strings (representing version values from multiple rows) as values
        # Therefore, each list of strings
        # concatenated with the corresponding key, recreates the original row
        # CONSTRAINT: version values must come first
        grouped_partitioned_data = group_data_by_feature_set(partitioned_data
                                   ,get_features=lambda x: "|+|".join(x[len(version_key):])
                                   # Combine feature values into a unique string identifier
                                   ,get_versions=lambda x:x[0:len(version_key)]
                                   ,logger=logger) # Extract header key values
        # The 'get_features' function combines the feature  values of each row
        # into a single string, acting as a unique identifier for a grouping.
        # The 'get_versions' function extracts the version values from each row.

        # grouped_partitioned_data = { 'B':
        # [['11.1', 'MongoDB', 'MongoDB 4.0', 'CentOS', 'CentOS 6'],
        # ['11.1', 'MongoDB', 'MongoDB 4.0', 'CentOS', 'CentOS 7'],
        # ['11.1', 'MongoDB', 'MongoDB 4.0', 'CentOS', 'CentOS 8'],
        # ['11.2', 'MongoDB', 'MongoDB 4.2', 'CentOS', 'CentOS 6'],
        # ['11.2', 'MongoDB', 'MongoDB 4.2', 'CentOS', 'CentOS 7'],
        # ['11.2', 'MongoDB', 'MongoDB 4.2', 'CentOS', 'CentOS 8'],
        # ['11.2', 'MongoDB', 'MongoDB 4.4', 'CentOS', 'CentOS 6'],
        # ['11.2', 'MongoDB', 'MongoDB 4.4', 'CentOS', 'CentOS 7'],
        # ['11.2', 'MongoDB', 'MongoDB 4.4', 'CentOS', 'CentOS 8'],
        # ['11.3', 'MongoDB', 'MongoDB 4.6', 'CentOS', 'CentOS 8'],
        # ['11.4', 'MongoDB', 'MongoDB 4.6', 'Windows', 'Windows 10']]
        # }
```

We will group *partitioned_data* based on values in the *feature_headers* columns. Since, the rows in *partitioned_data* have the same value for column under "Feature", *grouped_partitioned_data* will only have one key,value pair.

```
        # Loop thru grouped_partitioned_data dictionary
        # Each key represents one rows' feature value(s)
        # Each value represents version values from multiple rows
        for set_of_feature,version_data in grouped_partitioned_data.items():

            # Returns a consolidate/compressed representation
            # of the version values from multiple rows
            consolidated_version_rows = cartesian_decomposition(
                                                            version_data,
                                                            version_key,logger,
                                                            input_csv_path, partitioned_data)

            # version_data =
            # [['11.1', 'MongoDB', 'MongoDB 4.0', 'CentOS', 'CentOS 6'],
            # ['11.1', 'MongoDB', 'MongoDB 4.0', 'CentOS', 'CentOS 7'],
            # ['11.1', 'MongoDB', 'MongoDB 4.0', 'CentOS', 'CentOS 8'],
            # ['11.2', 'MongoDB', 'MongoDB 4.2', 'CentOS', 'CentOS 6'],
            # ['11.2', 'MongoDB', 'MongoDB 4.2', 'CentOS', 'CentOS 7'],
            # ['11.2', 'MongoDB', 'MongoDB 4.2', 'CentOS', 'CentOS 8'],
            # ['11.2', 'MongoDB', 'MongoDB 4.4', 'CentOS', 'CentOS 6'],
            # ['11.2', 'MongoDB', 'MongoDB 4.4', 'CentOS', 'CentOS 7'],
            # ['11.2', 'MongoDB', 'MongoDB 4.4', 'CentOS', 'CentOS 8'],
            # ['11.3', 'MongoDB', 'MongoDB 4.6', 'CentOS', 'CentOS 8'],
            # ['11.4', 'MongoDB', 'MongoDB 4.6', 'Windows', 'Windows 10']]
```

```
# -------------------- Inside cartesian_decomposition function -------------------
    # Getting all uniq vals per each column from partitioned data
    uniq_column_vals_part_data = get_uniq_vals_for_each_column(key_,partitioned_data)
    logger.debug("Stored all uniq vals in each column from part. data")
    # Returns a dict, where key is the column name,
    # and value is a list of unique vals in that column

    # Getting all uniq vals per each column from version data
    uniq_column_vals_version_data = get_uniq_vals_for_each_column(key_,version_data)
    logger.debug("Stored all uniq vals in each column from version data")

    # uniq_column_vals_part_data = {
    # 'GuardiumVersion': ['11.1', '11.2', '11.3', '11.4'],
    # 'DatabaseName': ['MongoDB'],
    # 'DatabaseVersion': ['MongoDB 4.0', 'MongoDB 4.2', 'MongoDB 4.4', 'MongoDB 4.6'],
    # 'OSName': ['CentOS', 'Windows'],
    # 'OSVersion': ['CentOS 6', 'CentOS 7', 'CentOS 8', 'Windows 10']}
```

Get all unique values in each column in a dictionary.

```

    # Generate all possible relevant ranges (ordered subsets)
    # per each column using uniq vals (Can blow up computationally)
    # eg. find_ranges([1,2,3]) = [[1],[2],[3],[1,2],[2,3],[1,2,3]]

    all_ranges = []
    for x in key_:
        ranges = find_relevant_ranges(list(uniq_column_vals_version_data[x])
                                  ,list(uniq_column_vals_part_data[x]))
        all_ranges.append(ranges)
    logger.debug("Generated all possible ordered set of uniq vals of each column")


    # all_ranges = [
    # [['11.1'], ['11.1', '11.2'], ['11.1', '11.2', '11.3'], ['11.1', '11.2', '11.3', '11.4'], ['11.2'], ['11.2',
        '11.3'], ['11.2', '11.3', '11.4'], ['11.3'], ['11.3', '11.4'], ['11.4']],
    # [['MongoDB']],
    # [['CentOS'], ['CentOS', 'Windows'], ['Windows']],
    # [['CentOS 6'], ['CentOS 6', 'CentOS 7'], ['CentOS 6', 'CentOS 7', 'CentOS 8'],
    # ['CentOS 6', 'CentOS 7', 'CentOS 8', 'Windows 10'], ['CentOS 7'], ['CentOS
    # 7', 'CentOS 8'], ['CentOS 7', 'CentOS 8', 'Windows 10'], ['CentOS 8'],
    # ['CentOS 8', 'Windows 10'], ['Windows 10']]
    ]
```

Gets each possible range for each column.

```
    # Generate all possible combinations using ranges from each column
    # Take cartesian product of each list of ranges
    # combinations([ [[1],[2],[3],[1,2],[2,3],[1,2,3]] , [["a"],["b"],["a","b"]] ]) =
    #     [
    #         [[1], ['a']],        [[1], ['b']],        [[1], ['a', 'b']],
    #         [[2], ['a']],        [[2], ['b']],        [[2], ['a', 'b']],
    #           [[3], ['a']],        [[3], ['b']],        [[3], ['a', 'b']],
    #           [[1, 2], ['a']],     [[1, 2], ['b']],     [[1, 2], ['a', 'b']],
    #           [[2, 3], ['a']],     [[2, 3], ['b']],     [[2, 3], ['a', 'b']],
    #           [[1, 2, 3], ['a']], [[1, 2, 3], ['b']], [[1, 2, 3], ['a', 'b']]
    #                                                                          ]
    # (Can blow up computationally)
    # Save each combination as a Combination Object
    combinations_list = []
    y = combinations(all_ranges)
    for _,x in enumerate(y):
        combinations_list.append(Combination(x))
    logger.debug("Generated all possible combinations of uniq val ordered sets of each column")
    # y = [
    # [['11.1'], ['MongoDB'], ['MongoDB 4.0'], ['CentOS'], ['CentOS 6']],
    # [['11.1'], ['MongoDB'], ['MongoDB 4.0'], ['CentOS'], ['CentOS 6', 'CentOS 7']],
    # [['11.1'], ['MongoDB'], ['MongoDB 4.0'], ['CentOS'], ['CentOS 6', 'CentOS 7', 'CentOS 8', 'Windows 10']],
    # ....,
    # [['11.4'], ['MongoDB'], ['MongoDB 4.6'], ['Windows'], ['Windows 10']]
    # ]
    # len(y) = 3000
```

Using each possible range from each column, generate all possible combinations.

```
    # Check if each row of version_data is compatible with any possible combination
    for _,row in enumerate(version_data):
        for _,combo in enumerate(combinations_list):
            # Check if row is compatible with combo (Cartesian product of combo includes row)
            # i.e row [1,a] is compatible with combo([[1], ['a', 'b']])
            # but row [1,a] is not compatible with combo([[2, 3], ['a', 'b']])
            if combo.combo_allows_row(row):
                combo.add_row(row) # Store all compatible rows for each combo
    logger.debug("Referenced all data rows with all possible combinations")

    # Sort combination from higher to lower capacity
    combinations_list = sorted(combinations_list, key=lambda x: -x.capacity)
    # combinations_list = [
    # Combination(length=5, key=[['11.1', '11.2', '11.3', '11.4'], ['MongoDB'],
    # ['MongoDB 4.0', 'MongoDB 4.2', 'MongoDB 4.4', 'MongoDB 4.6'], ['CentOS',
    # 'Windows'], ['CentOS 6', 'CentOS 7', 'CentOS 8', 'Windows 10']],
    # capacity=128, flow=11, rows=[....]),
    # Combination(length=5, key=[['11.1', '11.2', '11.3'], ['MongoDB'], ['MongoDB # 4.0', 'MongoDB 4.2', 'MongoDB
         4.4', 'MongoDB 4.6'], ['CentOS', 'Windows'], # ['CentOS 6', 'CentOS 7', 'CentOS 8', 'Windows 10']],
        capacity=96, flow=10, # rows=[...]),
    ...,
    # Combination(length=5, key=[['11.4'], ['MongoDB'], ['MongoDB 4.6'], ['Windows'], ['Windows 10']], capacity
        =1, flow=1, rows=[...])
    ]
```

Check each *row* in *version_data* against each of the 3000 *combo* in *combinations_list*. For example, the *row* = ['11.1', 'MongoDB', 'MongoDB 4.0', 'CentOS', 'CentOS 6'] is compatible with *Combination(key=[['11.1'], ['MongoDB'], ['MongoDB 4.0'], ['CentOS'], ['CentOS 6']])* and *Combination(key=[['11.1', '11.2', '11.3', '11.4'], ['MongoDB'], ['MongoDB 4.0', 'MongoDB 4.2', 'MongoDB 4.4', 'MongoDB 4.6'], ['CentOS', 'Windows'], ['CentOS 6', 'CentOS 7', 'CentOS 8', 'Windows 10']])*. But, is not compatible with *Combination(key=[['11.1', '11.2', '11.3', '11.4'], ['MongoDB'], ['MongoDB 4.2', 'MongoDB 4.4', 'MongoDB 4.6'], ['CentOS', 'Windows'], ['CentOS 6', 'CentOS 7', 'CentOS 8', 'Windows 10']])*. We will track these compatible rows in the *row* property of the *Combination* object.

```
    # Make copy of version_data
    version_data_copy = version_data

    # This will represent those combinations which represent exactly all rows of data
    final_combos = []

    # Loop thru each full capped combination and remove those lines from version_data_copy,
    # that can be represented by that combination
    # Continue till you have a list of combination that can
    # represent all lines/rows in version_data
    while len(version_data_copy) != 0:
        for _,combo in enumerate(combinations_list):
            if combo.is_full_cap(): # A combo is full_cap if len(combo.rows) == combo.capacity

                # if all combo.rows are present in version_data_copy, remove these rows
                # if not all combo.rows are present in version_data_copy, keep unchanged
                version_data_copy_filtered = remove_if_all_present(version_data_copy,combo.rows)

                if len(version_data_copy_filtered) != len(version_data_copy):
                    # if version_data_copy_filtered was changed
                    version_data_copy = version_data_copy_filtered
                    combinations_list.remove(combo)
                    #Check for subcombos
                    final_combos.append(combo)

    #
    return [final_combo.key for final_combo in final_combos]

# returns [
# [['11.2'], ['MongoDB'], ['MongoDB 4.2', 'MongoDB 4.4'], ['CentOS'], ['CentOS 6', 'CentOS 7', 'CentOS 8']]
# [['11.1'], ['MongoDB'], ['MongoDB 4.0'], ['CentOS'], ['CentOS 6', 'CentOS 7', 'CentOS 8']]
# [['11.3'], ['MongoDB'], ['MongoDB 4.6'], ['CentOS'], ['CentOS 8']]
# [['11.4'], ['MongoDB'], ['MongoDB 4.6'], ['Windows'], ['Windows 10']]
# ]
# version_data =
# [['11.1', 'MongoDB', 'MongoDB 4.0', 'CentOS', 'CentOS 6'],
# ['11.1', 'MongoDB', 'MongoDB 4.0', 'CentOS', 'CentOS 7'],
# ['11.1', 'MongoDB', 'MongoDB 4.0', 'CentOS', 'CentOS 8'],
# ['11.2', 'MongoDB', 'MongoDB 4.2', 'CentOS', 'CentOS 6'],
# ['11.2', 'MongoDB', 'MongoDB 4.2', 'CentOS', 'CentOS 7'],
# ['11.2', 'MongoDB', 'MongoDB 4.2', 'CentOS', 'CentOS 8'],
# ['11.2', 'MongoDB', 'MongoDB 4.4', 'CentOS', 'CentOS 6'],
# ['11.2', 'MongoDB', 'MongoDB 4.4', 'CentOS', 'CentOS 7'],
# ['11.2', 'MongoDB', 'MongoDB 4.4', 'CentOS', 'CentOS 8'],
# ['11.3', 'MongoDB', 'MongoDB 4.6', 'CentOS', 'CentOS 8'],
# ['11.4', 'MongoDB', 'MongoDB 4.6', 'Windows', 'Windows 10']]
```

Here we loop through each *combo* and a create a set of them, such that they represent exactly all rows of *version_data*

We continue this for all *uniq_vals* to create Table 2.