



**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

CZ4123 - Big Data Management

Group Project Report on Column Store Implementation

| Name | Matriculation Number |
|--------------------|-----------------------------|
| Goh Zheng Yang | U2020170H |
| Wang Xin Yan Lloyd | U2022916C |
| Fu Yongding | U1921155E |

1. Data Storage

1.1. Column store

```
def split_columns(data_file: str, zone_maps: Dict) -> None:
    """Splits the large csv into individual columns in their own files"""
    columns = get_columns(data_file=data_file)
    recreate_folders(folders=[SPLIT_DATA_FOLDER])
    opened_files = []
    with open(data_file, 'r') as f:
        next(f)
        min_max_dict = initialize_min_max_dict(zone_maps=zone_maps)
        curr_zone = 0
        i = 0
        for line in f:
            if i % MAX_FILE_LINE == MAX_FILE_LINE - 1:
                for col in min_max_dict:
                    min_max_dict[col]['max_idx'] = i
            i += 1
```

Figure 1: Implementation of column store database (full implementation in main.py)

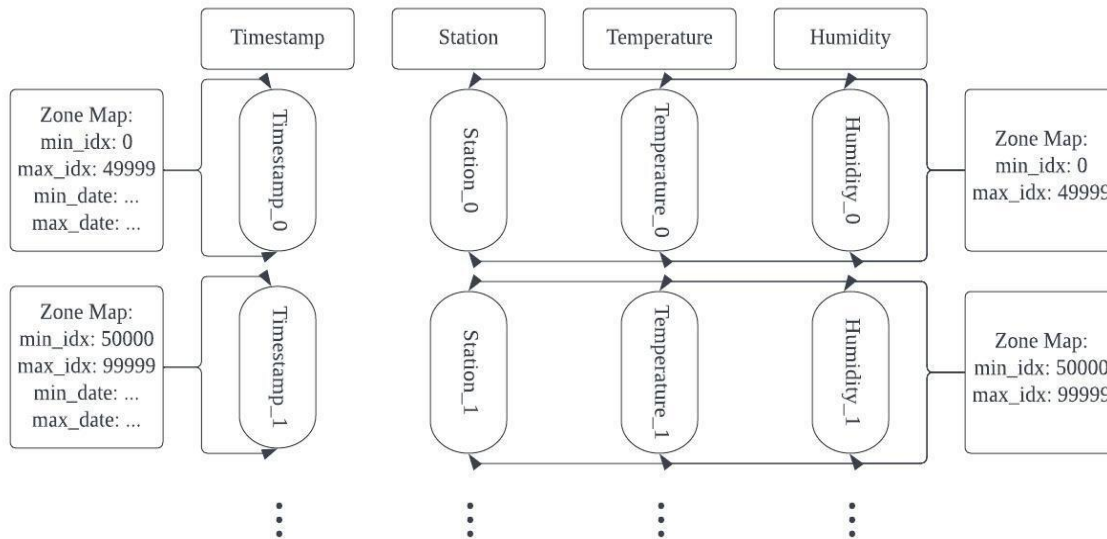


Figure 2: Depiction of column store database

The column store was implemented using the `split_columns` function, as seen in figure 1. Considering the situation when the data gets too big, we have also decided to split the data within columns. The dataset is first split into multiple columns, then each column is split into multiple smaller files using the variable `MAX_FILE_LINE` in `project_config.py`. When reading files, the file object created in python is actually a lazy generator of lines, so each iteration will only require the memory space of each line, hence minimising the memory used.

In this case, we shall term them as “zones” rather than “vectors” as vectors will process corresponding vectors in other columns together, but ours will not. Instead, every smaller file of each column will have a dictionary of its ranges stored, similar to zone maps. Although we are aware that zones are typically only a few pages in size, we have decided to adopt its terminology for easier understanding. These zone maps contain mainly the minimum index and

maximum indexes of the tuples stored in the file as they are already stored in a sequential manner on disk. The timestamp files have two more keys, namely minimum and maximum dates, which allow for easier search later on. Figure 2 shows a depiction of this storage method and the zone map calculations.

```
MAPPER = {
  'Station': {
    'Changi': '0',
    'Paya Lebar': '1'
  }
}
```

Figure 3: Using a mapper to convert the station name to a single character

Furthermore, we have noticed that the station column only has 2 enumerations, namely Changi and Paya Lebar. This allows us to convert them to a single binary character (0 for Changi and 1 for Paya Lebar) to save space as shown in Figure 3.

1.2. Intermediate storage

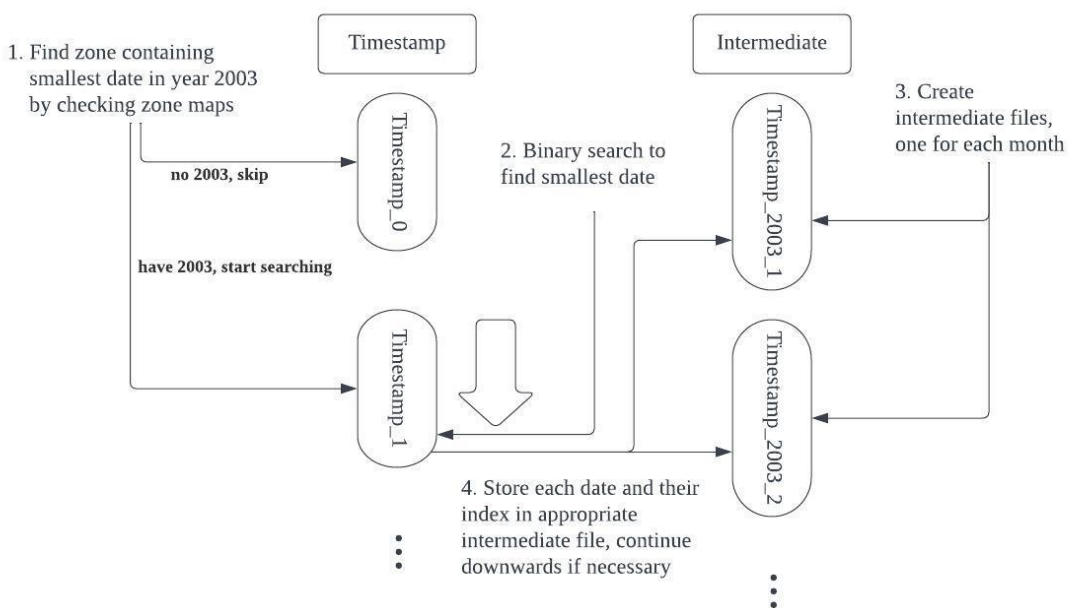


Figure 4: Example of intermediate storage for tuples in 2003

To simulate when intermediate files also get large, we have decided to also store the intermediate positions for each month in the query separately. Figure 4 outlines the steps taken to find and store the tuples with satisfying dates in their corresponding intermediate files. In

short, we first find the years that we are going to process, check the zone maps for the correct zone, enter the zone and store the tuples the files. The sorted nature of the timestamp column allows for binary searching the smallest date, hence speeding up the search process.

| archive > Timestamp > ≡ Timestamp_2003_1.txt | | |
|--|------------|-------|
| 1 | 2003-01-01 | 16928 |
| 2 | 2003-01-01 | 16929 |
| 3 | 2003-01-01 | 16930 |
| 4 | 2003-01-01 | 16931 |
| 5 | 2003-01-01 | 16932 |

Figure 5: Contents of intermediate file

The intermediate files store both the date and the index of the satisfying tuple as shown in figure 5 above. This allows us to reduce one more scan which we will explain in section 2.2.

2. Data Processing

2.1. Filtering tuples

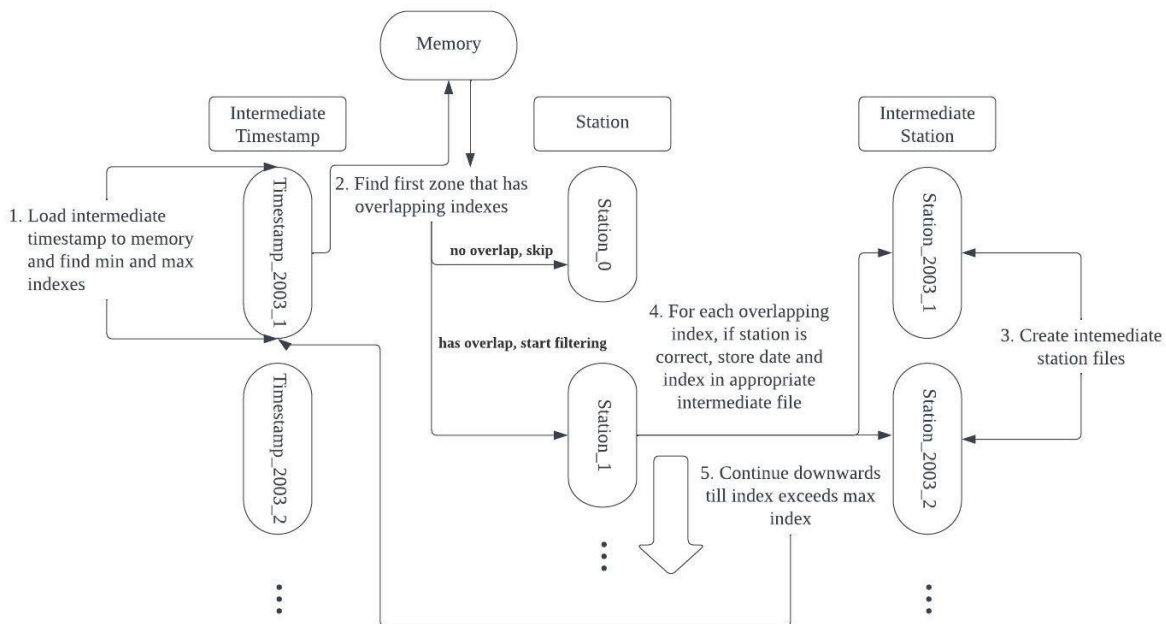


Figure 6: Filtering indexes from station data

The previous section in intermediate storage has already shown the steps taken to filter the timestamps. Figure 6 here outlines the steps taken to filter the stations. During the storage of the intermediate timestamp files, since we had iterated through a sorted timestamp column and stored the information, the intermediate file also contains sorted data. Hence, we can load it to memory and find the first row with the minimum index, and the last row with the maximum index.

We can then traverse through the zone maps of the station files to find the first zone that has overlapping indexes. Afterward, we check if the station is correct (1 or 0) and save it to the next intermediate file. This file also stores the same date and index of the satisfying tuple.

2.2. Shared scan

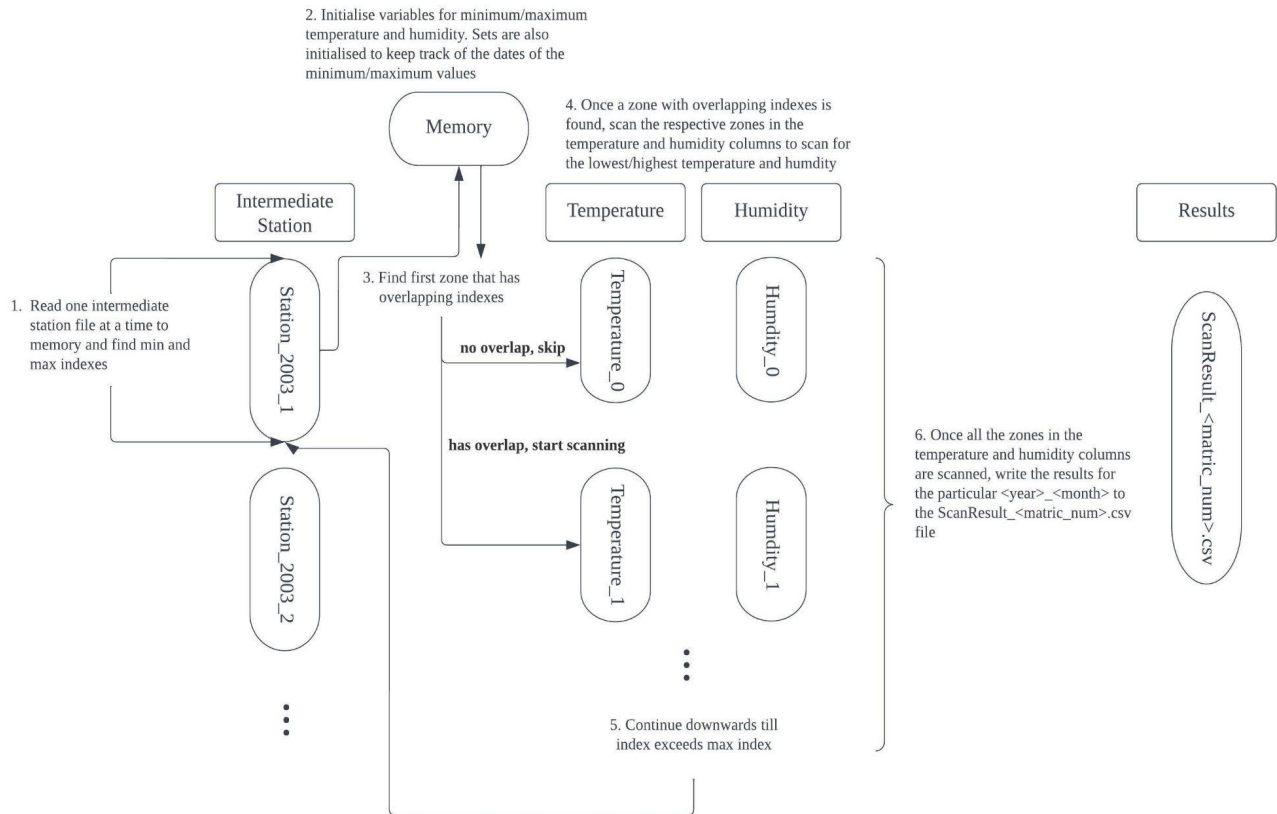


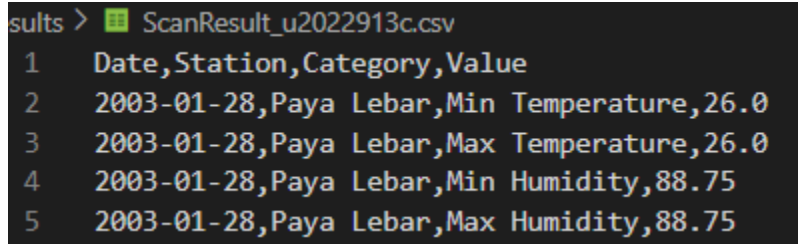
Figure 7: Scanning temperature and humidity columns

After scanning the station column and storing the intermediate results into a file that is stored on the disk, the program will process both the temperature and humidity column at the same time to find the monthly minimum and maximum values of temperature and humidity. Figure 7 outlines the steps taken to query the required values. In this stage, the intermediate station files, which are named as **Station_<year>_<month>**, are processed one at a time. If there are no entries in an intermediate station file, meaning there are no entries that satisfies the year, month, and station requirement, the program will move on to the next intermediate file to process. The first row in the intermediate file is the minimum index while the maximum index is the last row. The minimum and maximum index is used to traverse through the zone maps of the temperature and humidity columns. Once a zone with overlapping indexes are found, both the zone in the temperature and humidity column will be scanned to search for the required values. Some rows in the temperature and humidity column have the value “M”. These rows are not taken into consideration when scanning for the required results. After going through all

the zones, the results for the particular month will be appended to the **ScanResult_<matric_num>.csv** file. After scanning all the intermediate station file, all the required results will be obtained and written to the results file.

3. Experiment Result

3.1. Output

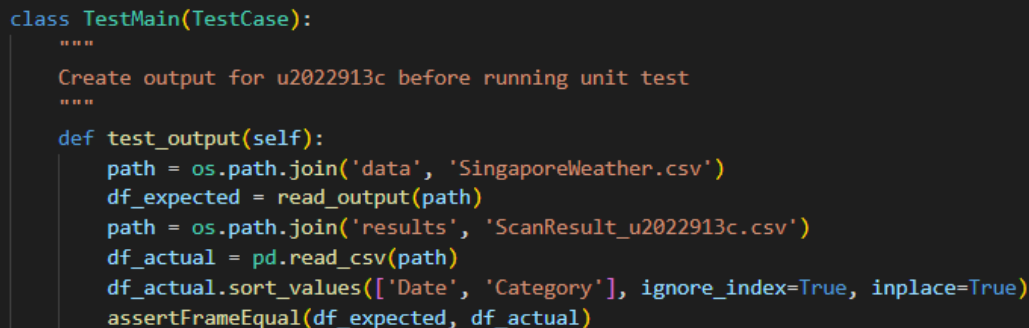


```
results > ScanResult_u2022913c.csv
1 Date,Station,Category,Value
2 2003-01-28,Paya Lebar,Min Temperature,26.0
3 2003-01-28,Paya Lebar,Max Temperature,26.0
4 2003-01-28,Paya Lebar,Min Humidity,88.75
5 2003-01-28,Paya Lebar,Max Humidity,88.75
```

Figure 8: Sample output with input ending with 13 as last 2 digits

Figure 8 shows a snippet of the output for the matriculation number u2022913c. The results of the queries can be found in the results folder. In this case, the queried station and years should be Paya Lebar and 2003 and 2013. Our output seems to filter the checks accurately. It is not necessarily sequenced in order, and it stores multiple tuples for every month if they have the same values for a particular category. This is in line with the requirements of the project.

3.2. Evaluation of correctness



```
class TestMain(TestCase):
    """
    Create output for u2022913c before running unit test
    """
    def test_output(self):
        path = os.path.join('data', 'SingaporeWeather.csv')
        df_expected = read_output(path)
        path = os.path.join('results', 'ScanResult_u2022913c.csv')
        df_actual = pd.read_csv(path)
        df_actual.sort_values(['Date', 'Category'], ignore_index=True, inplace=True)
        assertFrameEqual(df_expected, df_actual)
```

Figure 8: Test case for u2022913c

Figure 8 shows a snippet of the code used to test the output for the matriculation number u2022913c. Since the dataset size is rather small, we used Python Pandas to process the data (in a row-oriented manner) this time. It performs the same filtering and calculations on the main dataset and compares the results with the generated results in the results folder. By using the command provided in the README, the test case output shows that the result obtained is correct. The results are equivalent in both methods.

4. Contribution Form

| Name | Detailed Individual Contribution | Percentage |
|--------------------|----------------------------------|------------|
| Wang Xin Yan Lloyd | Code and report | 33.3% |
| Fu Yongding | Code and report | 33.3% |
| Goh Zheng Yang | Code and report | 33.3% |

Name and Signature of all members

A handwritten signature in grey ink, appearing to be 'WXY' with a stylized flourish at the end.

Name: Wang Xin Yan Lloyd

A handwritten signature in black ink, consisting of the letters 'Y' and 'D' separated by a dot.

Name: Fu Yongding

A handwritten signature in black ink, featuring a large, bold, stylized 'G' followed by a horizontal line.

Name: Goh Zheng Yang