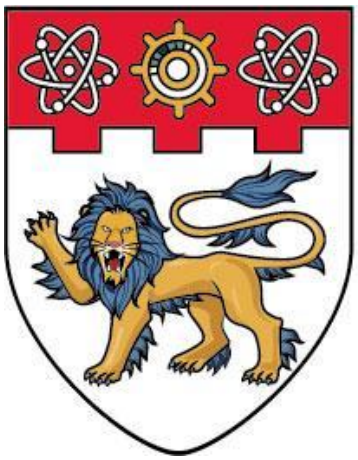


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CE/CZ4123 Big Data Management

PROJECT MILESTONE 1

Lee Kai Shern (U1820793J)

Table of Content

1 - Data Storage

This section will explain how the program handles and stores data in a column-store approach. Generally, all data will be stored and handled in bytes format. The decision of using bytes instead of storing it as raw data format is because most of the columns have fixed amount of bytes, with extra calculation steps, we can obtain the location of a page in memory and disk.

The basic unit of reading or writing will be a `Page`. The `Page` is implemented using an array of bytes with the remaining number of bytes as metadata. The length of the bytes array ① imitates the size of each page. Memory and disk will use the same size of page.

1.1 Main memory

The `storageSys` (memory) is just simply an array of `Page`'s. The length of `storageSys` ② imitates the amount of pages available on memory. The total memory (in Byte) available will be ① * ②. The words location, address and index will be used interchangeably, they all mean the index of the page in the memory.

The table below shows the variables in the `MemoryController` class.

Variable Name : Type	Description
numPage: *int*	Number of page ②
pageSize: *int*	Size of each page ①
reservedPage: *int*	The current number of reserved page
isReserved: *boolean[]*	Indicates whether the page at a particular index is being reserved (can not be rewritten until it is being released)
nextAddr: *int*	A pointer which points to the next writable page
pageTable_D2M: *Hashmap*	Hashmap storing the location in disk to location in memory
memory: *StorageSys*	Array of *Page*'s for storage purpose

The table below shows the functions in the `MemoryController` class which is responsible for all actions that involve the memory. Do note that only key functionalities are included in the table below.

Function name	Description	Input parameter	Return object

writeToMemory	Uses simple First-in-First -Out rule and excludes reserve page when moving the *nextAddr* pointer to the next writable page	● diskAddr: *int* ● pageToWrite: *Page* The disk address is solely for updating of page table, memory controller is not responsible to write to disk	● memoryAddr: *int* Return the index in memory of the page being written to
getPage	Read page at the input index	● memoryAddr: *int*	● page: Page Page is returned by reference, no new page is being created.
reserveMemory	Mark the page at *nextAddr* as reserved and not writable. The *nextAddr* pointer will be moved to next writable page	-	● memoryAddr: *int* Return the index in memory of the page being reserved
releaseMemory	Mark the page at input address to not reserved and become writable again	● memoryAddr: *int*	-

Even though the page in memory will be flushed to disk from time to time, the content of it will not be reset after the flush. The page table will still keep the memory address to disk address pair. The flushed page is still available in memory until the next time the *nextAddr* pointer points to it.

1.2 Disk

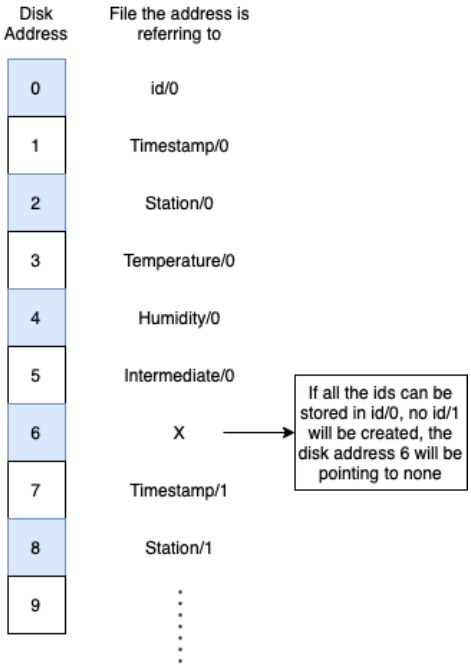
The disk is implemented differently , there is no `storageSys` with an array of Pages as memory. In contrast, each page is being stored as a byte file as such:

Each column is stored under a separated folder with multiple byte files. The folder *Intermediate* is used to store all the output (intermediate and final) to prevent situations where the program does not have enough memory to run the queries. Every page in memory that is full will be flushed to disk to become a new byte file, so that the page in memory can be released and reused.

A number will be assigned to each column eg: id:0, Timestamp:1, Station:2…Intermediate:5. The number will be used to derive the relative index in disk which should be unique for each byte file across different column folders.

*Disk address = order in folder * 6 + number assigned to the column*

For example for the file Timestamp/2, its disk address will be 2 * 6 + 1 = 13. The image below will help to visualize the above concept.



Do note that the concept of disk address is just a numbering system, there is no unused space on the disk and all the numbers are calculated only when the file is read to memory for any process.

The table below shows the key variables in the `DiskController` class:

Variable Name : Type	Description
pageSize: *int*	Size of each page ①
columnName: *String[]*	Array containing names of the columns arranged as the header of SingaporeWeather.csv
pageCounter: *int[]*	Array to keep track of the number of files created for each column
baseFilePath: *String*	File path to which the column-based byte files should be stored

There are only 2 functions in this controller class which will be used for queries - *read from file* and *write to file*. Other functions are for reading SingaporeWeather.csv to a column-based file at initialisation and will be discussed in the following subsection.

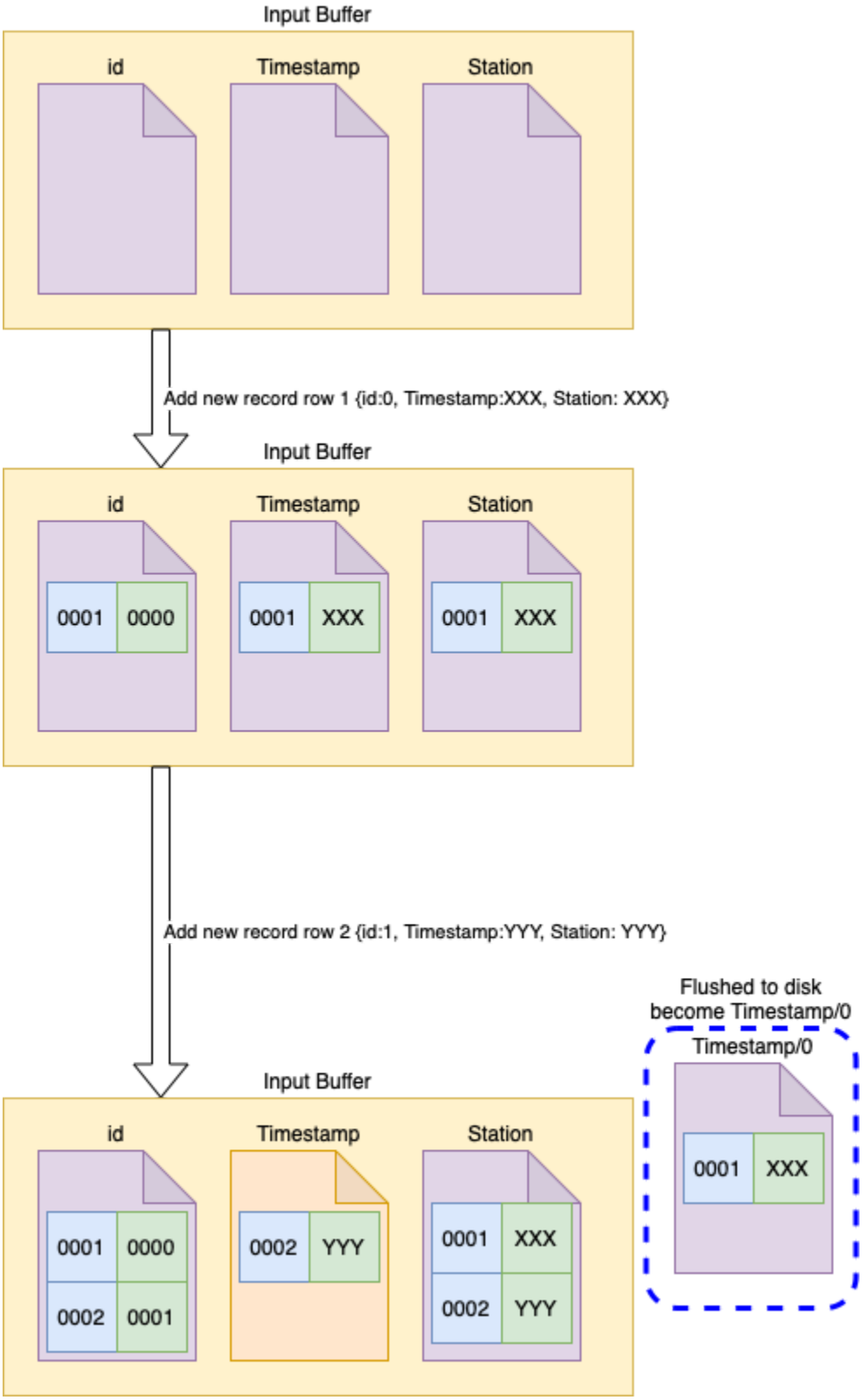
1.3. Input file reading (Initialisation)

1.3.1 Implementation

The initialisation is handled by the `DiskController` class.

1. Allocate 5 pages of memory (input buffer) at initialisation. (There are 5 columns in the data)
2. Read each row in the SingaporeWeather.csv file and split it into 5 elements. There is a counter indicating which row the reader is currently at; the row number will be added into the page together with the element.
3. For each of the elements, convert it to bytes array and append it to its allocated input buffer eg: id to page at index 0, timestamp to page at index 1 and so on
4. If any of the pages is unable to take in a new record, only the page itself will be flushed to disk (save as byte file), the other pages in input buffer will just append the record
5. Reset the flushed page to an empty page and append the unadded record from the previous step.
6. Continue until there is no more rows
7. Flushed all the pages in input buffer to disk

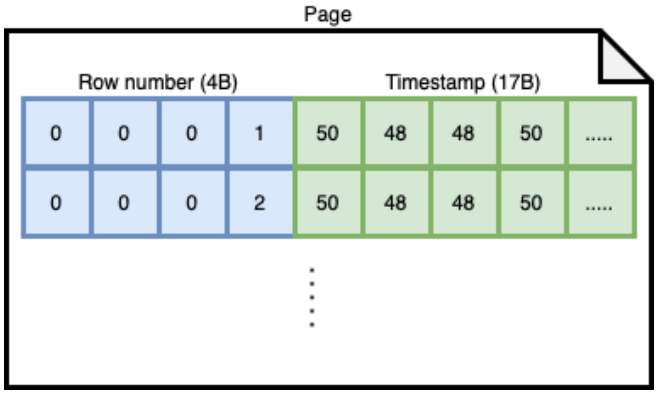
The diagram below visualises the steps of the initialisation:



1.3.2 Data format

Column	Treated as
id	Integer (4 bytes)
Timestamp	Fixed length string (17 bytes)
Station	Integer (4 bytes) - “Changi “ will be converted to 1 and “Paya Lebar” to 2, fixed length of bytes will make the query run more efficiently and require less storage space.
Humidity	Float (4 bytes)
Temperature	Float (4 bytes)

The image below shows part of a page of Timestamp:



1.3.3 Exception handling

For rows with ‘M’ as the humidity or temperature, the row will still be read but M will be converted to NaN in float, then to a bytes array. These rows are ignored when running the query.

1.4 Read and write of file

When the process wants to read a file, it must have the **disk address** as the input parameter; the disk address will then be translated into the corresponding folders and byte file index. The file will be read and the byte arrays will be written into a page in memory. The memory controller is responsible for updating the page table with the location in disk and memory.

The write-to-file discussed here is only for the process of writing intermediate results to the *Intermediate* folder. When the process wants to write to a file, it must have the **memory address** where the intermediate results are located. The byte array of the page will then be streamed and saved as a file with the file index obtained from `pageCounter`.

2 - Data Processing

2.1 Overview of query steps

Steps **(?) indicates red circled number in the image above**

1. Timestamp column is filtered, extracting those rows which match the years 2003 and 2013
2. Using the timestamp table from step 1, the station column is being retrieved and run the match. If it matches station 1 (Changi), it will be written to the intermediate station table (2). At the same time, the timestamp is being filtered again, leaving only those rows with matching stations (2).
3. Using the row number in (2), the humidity and temperature of the particular row are being retrieved.
4. The obtained value is being compared with the values in the reserved output buffer (3, 4). For the maximum case if it is larger than the recorded value, the stored value will be overwritten, and vice versa for the minimum case. The row numbers used here are relative to the previous step (dotted yellow box) instead of the original row number (blue box)
5. After checking all the satisfying rows in (2), the output buffers will be joined with the intermediate timestamp (2) and intermediate station (1) and be output to the ScanResult.csv file.

All the intermediate results are piggy-backed with the values e.g. row number + Timestamp. This is because these values will need to be displayed as the final result. Another option is just by passing the row number as intermediate results. However, at the last step of the query, there is a need to reload these pages into the memory to obtain the corresponding values.

2.2 Scanning of columns

The first step is to filter by year, as it has low selectivity. All the files (one by one) in the *timestamp* folder are being brought to memory to do the matching.

The scanning is done by having a pointer which moves by sizeOfData+4 (1.3.2 Data format). The value is then compared to the target and only the matching rows will be saved to the output buffer.

After obtaining the intermediate timestamp table, there is no need to scan through all files in the *station* folder. Using the row number, the location in disk of the intended column can be identified (+4 for storage of the row number):

```
#File to read = rowNumber // (pageSize//(size of data+4))
```

For example, reading of station of row number = 2345, pageSize = 1024

#File to read = 2345 // (1024// (4+4)) = 18

The file containing row number 2345 of the station column can be found at file *station/18*.
The reason to re-index at step 4 (2.1 Overview of query steps) is also to help in identifying the location of the intermediate files.

2.3 Finding the minimum and maximum values

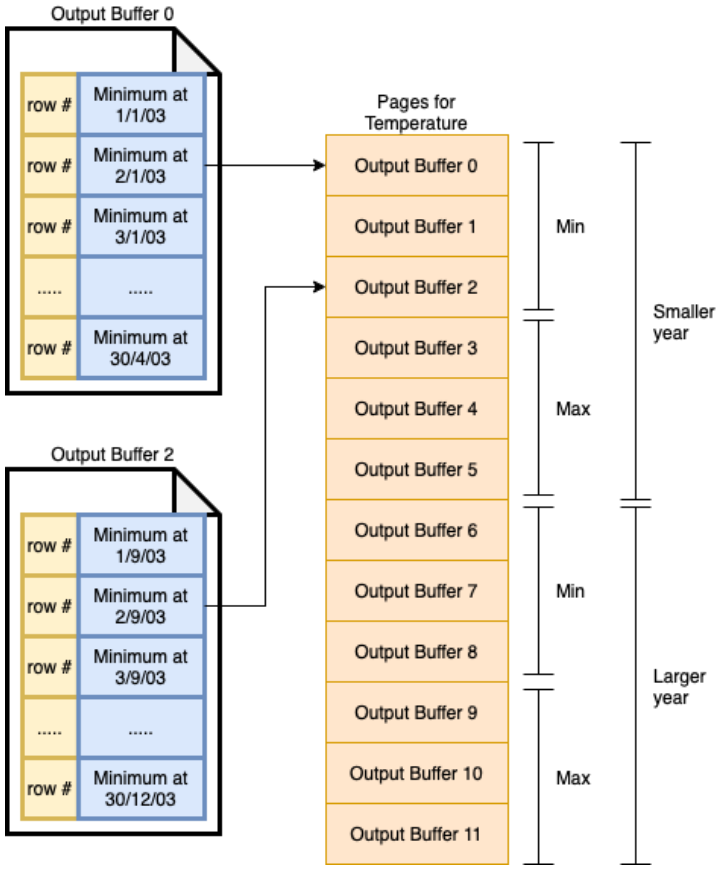
This step requires the most amount of memory. For all the filtering steps, it only requires a minimum of 2 pages, one for input and the other for output buffer. The output buffer can be immediately flushed to disk and be reused.

Number of pages required at this step =

```
int numMonthPerPage = (int) Math.floor((float) PAGESIZE / (31 * 8));
int numPagePerType = (int) Math.ceil((float) 12 / numMonthPerPage);
int numPageRequired = numPagePerType * 4 * 2; // 4 types 2 yrs
```

For the page size of 1024, the output buffers will need to be 24 pages. The higher requirement of memory is to filter the unique date value (excluding time in timestamp). In order to achieve that, all date values seen must be stored.

The method is being provisioned by the rule that each date can only be written to a specific location in a page. The image below will explain the idea more clearly, it is illustrated based on page size = 1024.

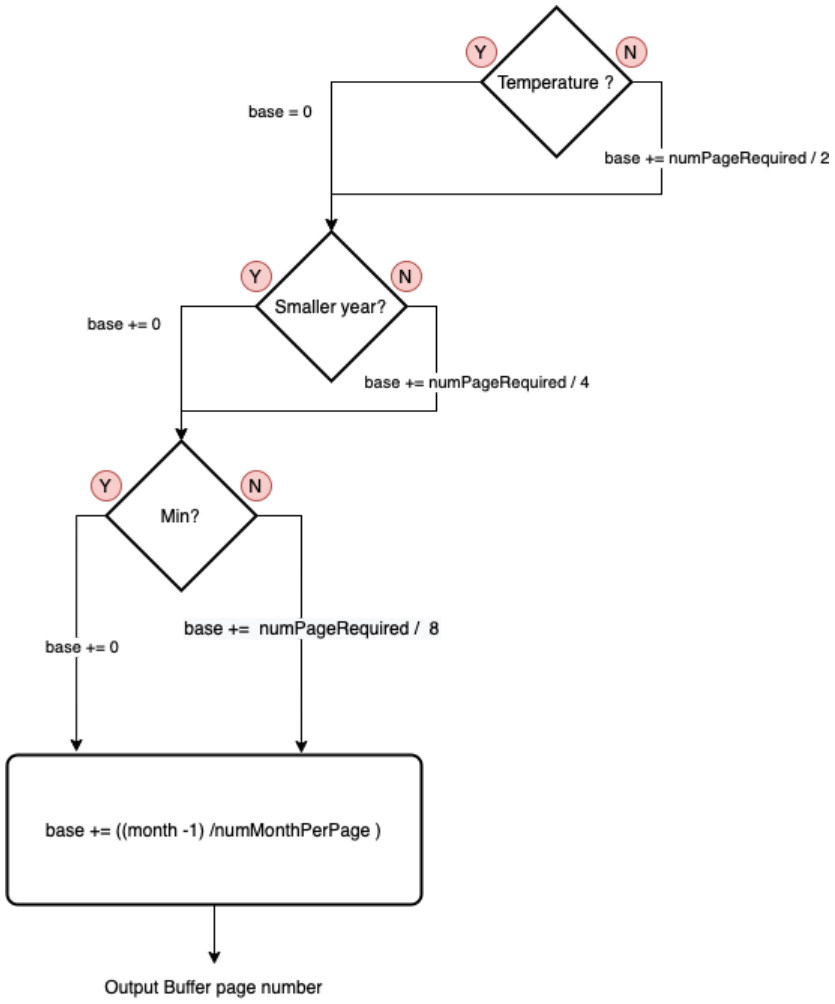


The value in Minimum at 1/1/03 to 31/1/03 has the same value.

Steps (to obtain minimum temperature)

1. Obtain the record value (current temperature)
2. Obtain the current min temperature by scanning the section of the month in the output buffer e.g.: 1-31 Jan (any value will do as they are all the same).
3. If the current min temperature > current temperature, all the values in the month will be cleared. The current temperature will be written to the location of its date.
4. If nothing is found in the month section or current temperature = current min temperature, current temperature will be written to the location of its date

The flowchart below illustrates how to obtain the page which the record should be compared to:



For example, to find the page where the larger year's June's maximum temperature should be written to: Using pageSize = 1024

Base = 0 + numPageRequired/4 + numPageRequired/8 + ((6 -1)//numMonthPerPage)

= 0 + 24/4 + 24/8 + 5//4

= 10

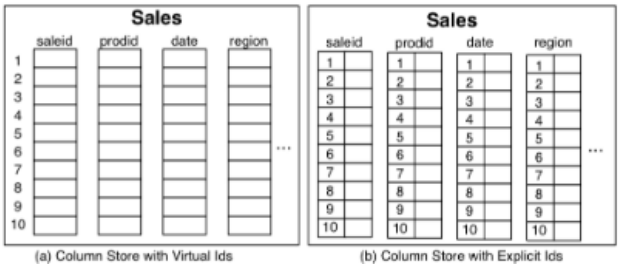
If the value is larger than the current maximum, it should be written to page 10 of the output buffer at index = ((month -1)//numMonthPerPage) * 31 *8 + (day -1) * 8.

After running all the rows in the input (intermediate timestamp), all the output buffer pages will be collated with empty values being ignored and output as byte files, leaving only those dates having the maximum or minimum value. And based on the index of the page being collated, the category column is being added.

Finally the program will retrieve the collated byte files and convert them into human readable string then output as a csv file.

2.4 How to improve efficiency

Throughout the process, all the values are tagged with a row number similar to the idea in (b) in the image below. This is for the cross-checking of row numbers when matching the rows from different columns and it makes debugging easier. However, since all the data stored are converted to fixed size, the idea in (a) would perform better with less space needed and still be able to find the desired page using the formula mentioned above.



Not only the location of the file can be identified, the start index in byte array (page) can also be calculated as such:

```
StartIndex = rowNumber % (pageSize//(size of data+4)) * (size of data+4)
```

For example, reading of station of row number = 2345, pageSize = 1024

#StartIndex = 2345 % (1024// (4+4)) * *(4+4)* = 328

The file containing row number 2345 of the station column can be found at *station/18*. And the starting index of it is the 328th byte. This allows direct access to the single record data compared to looping through all the bytes in the page. This method is only feasible because the page is implemented as an array of bytes which support direct access.

Instead of scanning through data of a month in the output buffer (31 rows per month) when looking for the current minimum and maximum, we could store the month min and max value on another page or buffer in memory and simply update and retrieve the value from there.

3 - Experiment Result

3.1 Result

The snippet below indicates the progress, with the last line indicating the location and filename which the final output is being written to.

```
Reading SingaporeWeather.csv to column stored files....
Running the query......
Output file generated at /Users/kaishern/Desktop/NTU/4123-Big Data Management/Lab/Milestone1_4123/src/main/resources/ScanResult.csv
```

The image below shows part of the final output file running for station **Paya Lebar** and the years of **2003** and **2013**.

```
Date,Station,Category,Value
2003-01-28,Paya Lebar,Min Temperature,26.0
2003-03-15,Paya Lebar,Min Temperature,30.0
2003-11-11,Paya Lebar,Min Temperature,25.0
2003-01-28,Paya Lebar,Max Temperature,26.0
2003-03-15,Paya Lebar,Max Temperature,30.0
2003-11-11,Paya Lebar,Max Temperature,25.0
2013-01-08,Paya Lebar,Min Temperature,23.0
2013-01-19,Paya Lebar,Min Temperature,23.0
2013-02-10,Paya Lebar,Min Temperature,23.0
2013-03-26,Paya Lebar,Min Temperature,24.0
2013-04-14,Paya Lebar,Min Temperature,23.0
2013-04-17,Paya Lebar,Min Temperature,23.0
2013-05-16,Paya Lebar,Min Temperature,24.0
2013-06-01,Paya Lebar,Min Temperature,24.0
2013-06-05,Paya Lebar,Min Temperature,24.0
2013-07-09,Paya Lebar,Min Temperature,24.0
2013-07-27,Paya Lebar,Min Temperature,24.0
2013-08-01,Paya Lebar,Min Temperature,23.0
2013-08-11,Paya Lebar,Min Temperature,23.0
2013-08-13,Paya Lebar,Min Temperature,23.0
2013-09-07,Paya Lebar,Min Temperature,23.0
2013-09-08,Paya Lebar,Min Temperature,23.0
2013-09-25,Paya Lebar,Min Temperature,23.0
2013-09-29,Paya Lebar,Min Temperature,23.0
2013-10-15,Paya Lebar,Min Temperature,23.0
2013-10-20,Paya Lebar,Min Temperature,23.0
2013-10-28,Paya Lebar,Min Temperature,23.0
2013-11-12,Paya Lebar,Min Temperature,23.0
2013-11-13,Paya Lebar,Min Temperature,23.0
2013-11-14,Paya Lebar,Min Temperature,23.0
2013-11-23,Paya Lebar,Min Temperature,23.0
2013-12-18,Paya Lebar,Min Temperature,22.0
2013-01-07,Paya Lebar,Max Temperature,35.0
2013-02-23,Paya Lebar,Max Temperature,34.0
```

3.2 Result Verification

Using SQLite, the two outputs are being cross-checked.
Following is the SQL code is used to achieve the same purpose.

```
CREATE TABLE sqlOutput AS
WITH tempT as
  (SELECT SUBSTR(Timestamp,0, 8) month, station, MAX(humidity) maxH, MAX(temperature) maxT, MIN(humidity)
minH, MIN(temperature) minT
    FROM SingaporeWeather_1
    WHERE (timestamp LIKE "2003%" OR timestamp LIKE "2013%") AND station = "Paya Lebar"
      GROUP BY month)
SELECT DISTINCT(SUBSTR(I.Timestamp,0, 11)) as "Date", I.Station,
            "Min Humidity    " as Category, I.Humidity as value
            FROM SingaporeWeather_1 as I JOIN tempT as r on I.humidity = r.minH
            AND SUBSTR(I.Timestamp,0, 8) = r.month AND I.Station = r.Station
UNION
SELECT DISTINCT(SUBSTR(I.Timestamp,0, 11)) as "Date", I.Station,
             "Max Humidity    " as Category, I.Humidity as value
             FROM SingaporeWeather_1 as I JOIN tempT as r on I.humidity = r.maxH
             AND SUBSTR(I.Timestamp,0, 8) = r.month AND I.Station = r.Station
UNION
SELECT DISTINCT(SUBSTR(I.Timestamp,0, 11)) as "Date", I.Station,
            "Min Temperature" as Category, I.Temperature as value
             FROM SingaporeWeather_1 as I JOIN tempT as r on I.temperature = r.minT
            AND SUBSTR(I.Timestamp,0, 8) = r.month AND I.Station = r.Station
UNION
SELECT DISTINCT(SUBSTR(I.Timestamp,0, 11)) as "Date", I.Station,
            "Max Temperature" as Category, I.Temperature as value
            FROM SingaporeWeather_1 as I JOIN tempT as r on I.temperature = r.maxT
            AND SUBSTR(I.Timestamp,0, 8) = r.month AND I.Station = r.Station
ORDER BY 2 DESC;
```

The ScanResult.csv is converted to a SQL table and joined with the sqlOutput table using the following SQL code to ensure that all rows exist in the other table.

```
SELECT * FROM ScanResult as I
  LEFT JOIN sqlOutput sO on I.Date = sO.Date
     AND rtrim(I.Category) = rtrim(sO.Category) AND I.value = sO.value;
```

The rtrim is to remove padded space to ensure the category has the same length.

All the rows in ScanResult find their exact rows in the sqlOutput table.

Date	Station	Category	Value	Date	Station	Category	value
2003-01-28	Paya Lebar	Min Temperature	26	2003-01-28	Paya Lebar	Min Temperature	26
2003-03-15	Paya Lebar	Min Temperature	30	2003-03-15	Paya Lebar	Min Temperature	30
2003-11-11	Paya Lebar	Min Temperature	25	2003-11-11	Paya Lebar	Min Temperature	25
2003-01-28	Paya Lebar	Max Temperature	26	2003-01-28	Paya Lebar	Max Temperature	26
2003-03-15	Paya Lebar	Max Temperature	30	2003-03-15	Paya Lebar	Max Temperature	30
2003-11-11	Paya Lebar	Max Temperature	25	2003-11-11	Paya Lebar	Max Temperature	25
2013-01-08	Paya Lebar	Min Temperature	23	2013-01-08	Paya Lebar	Min Temperature	23
2013-01-19	Paya Lebar	Min Temperature	23	2013-01-19	Paya Lebar	Min Temperature	23
2013-02-10	Paya Lebar	Min Temperature	23	2013-02-10	Paya Lebar	Min Temperature	23
2013-03-26	Paya Lebar	Min Temperature	24	2013-03-26	Paya Lebar	Min Temperature	24
2013-04-14	Paya Lebar	Min Temperature	23	2013-04-14	Paya Lebar	Min Temperature	23
2013-04-17	Paya Lebar	Min Temperature	23	2013-04-17	Paya Lebar	Min Temperature	23
2013-05-16	Paya Lebar	Min Temperature	24	2013-05-16	Paya Lebar	Min Temperature	24
2013-06-01	Paya Lebar	Min Temperature	24	2013-06-01	Paya Lebar	Min Temperature	24
2013-06-05	Paya Lebar	Min Temperature	24	2013-06-05	Paya Lebar	Min Temperature	24
2013-07-09	Paya Lebar	Min Temperature	24	2013-07-09	Paya Lebar	Min Temperature	24
2013-07-27	Paya Lebar	Min Temperature	24	2013-07-27	Paya Lebar	Min Temperature	24
2013-08-01	Paya Lebar	Min Temperature	23	2013-08-01	Paya Lebar	Min Temperature	23
2013-08-11	Paya Lebar	Min Temperature	23	2013-08-11	Paya Lebar	Min Temperature	23
2013-08-13	Paya Lebar	Min Temperature	23	2013-08-13	Paya Lebar	Min Temperature	23
2013-09-07	Paya Lebar	Min Temperature	23	2013-09-07	Paya Lebar	Min Temperature	23
2013-09-08	Paya Lebar	Min Temperature	23	2013-09-08	Paya Lebar	Min Temperature	23
2013-09-25	Paya Lebar	Min Temperature	23	2013-09-25	Paya Lebar	Min Temperature	23
2013-09-29	Paya Lebar	Min Temperature	23	2013-09-29	Paya Lebar	Min Temperature	23
2013-10-15	Paya Lebar	Min Temperature	23	2013-10-15	Paya Lebar	Min Temperature	23
2013-10-20	Paya Lebar	Min Temperature	23	2013-10-20	Paya Lebar	Min Temperature	23
2013-10-28	Paya Lebar	Min Temperature	23	2013-10-28	Paya Lebar	Min Temperature	23
2013-11-12	Paya Lebar	Min Temperature	23	2013-11-12	Paya Lebar	Min Temperature	23
2013-11-13	Paya Lebar	Min Temperature	23	2013-11-13	Paya Lebar	Min Temperature	23