HOMEWORK 2 REPORT

Bui Tien Cuong
2017 - 20722

1. Hardware specification
    a. OS: Ubuntu 16.04 x64
    b. Ram memory: 8Gb
    c. CPU: Intel(R) Core(TM) i5-6600 CPU @ 3.30GHz
    d. HDD: 256Gb
    e. C/C++ compiler: g++ 5.4.0
2. Column Table implementation

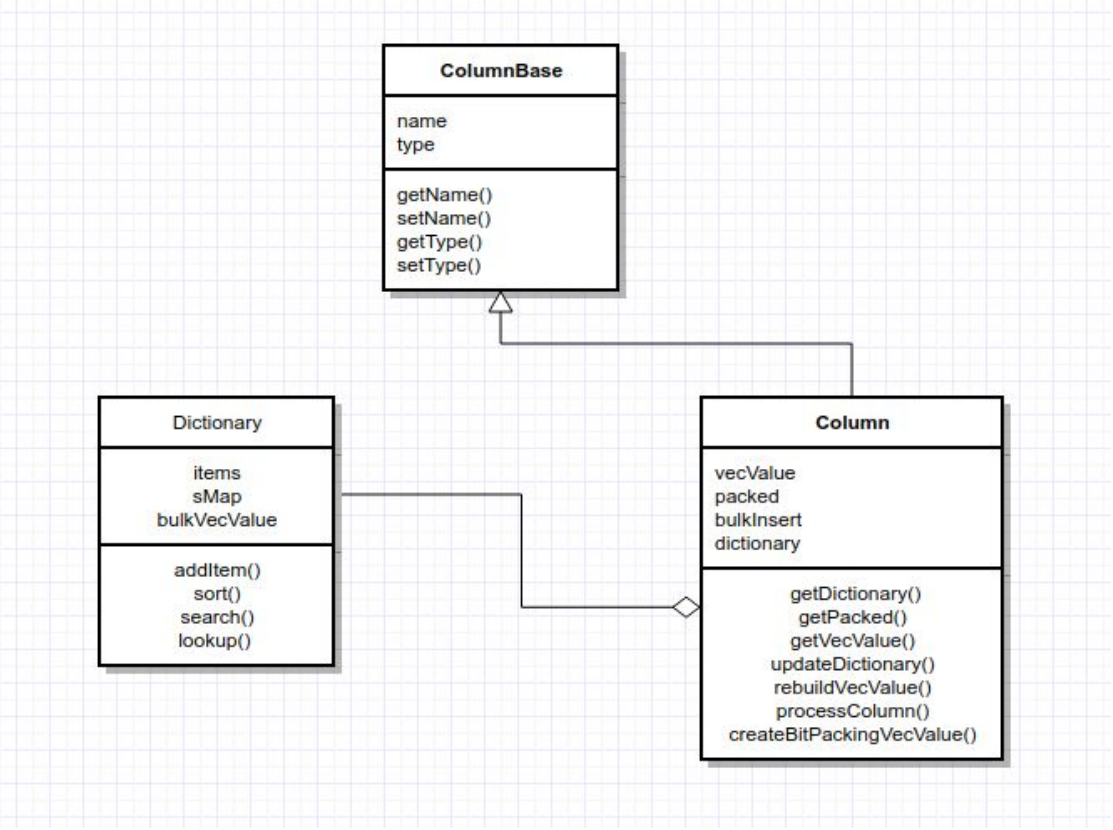

Figure 1: Class diagram

a. ColumeBase

- ColumnBase is an abstract class which contains name and type properties.
- ColumnBase.h contains declaration of methods and properties
- ColumnBase.cpp contains implementation code of get and set methods for name and type.

b. Column

- Column class is a template class, and extends from ColumnBase class. Using template is a efficient way to init a column with corresponding types (int, long, bigint, string).
- Column.h: Contains implementation code of properties and methods.
- Properties

Name	Type	Description
dictionary	Dictionary<T>	contain distinct values of a column
packed	PackedArray	contain bit-packed values of a column
vecValue	vector<size_t>	contain the position of each cell in dictionary
bulkInsert	bool	A Flag is to check whether sort dictionary after finish creating

- Method

Name	Type	Description
getDictionary	Dictionary<T>*	contain distinct values of a column
getPacked	PackedArray*	contain bit-packed values of a column
getVecValue	vector<size_t>*	contain the position of each cell in dictionary
updateDictionary	void	add item to the dictionary
rebuildVecValue	void	sort dictionary after finish inserting
processColumn	void	sort dictionary after finish inserting and create bit-packed array
createBitPackingVecValue	void	Create bit-packed column from vecValue
printVecValue	void	Print vecValue after bit-packed processing

c. Dictionary

- Dictionary class contains code for adding items, sorting, and searching
- Dictionary.h: declares properties and methods
- Dictionary.cpp: implements codes
- Properties:

Name	Type	Description
items	vector<T>	contain distinct values of a column
sMap	unordered_map	Temp map uses in adding process to verify unique property
bulkVecValue	vector<size_t>	contain regular positions of value in dictionary during adding step, is to re-order the right position of vecValue array and has been removed after sorting step

- Methods:

Name	Type	Description
lookup	T	return an item corresponding to input index
addItem	void	Add new item to dictionary, skip one if it is already existed
search	void	search items with conditions
getMemoryConsumption	int	get memory size of items array

3. Results

Memory is the physical consumption memory of current process. This is the result when process file “sample-game.csv” with size 88.7 Mb.

- Loading status

Memory status before loading: 1740 kb
Loading time: 7.047s
Memory consumption after loading: 137640 kb

- sid = 40

Dictionary size of sid: 41 with size: 172 bytes
Total row of sid = 40: 1
Running time of first expression is: 4.5e-05s
Query result: 11 is the position of sid = 40 in column dictionary
Dictionary[11] = 40
Memory consumption after perform ‘sid = 40’: 137640 kb

- V > 5,000,000

Dictionary size of v : 387008 with size: 1548040
Total row of V > 5,000,000: 220
Running time of V > 5,000,000: 0.002686s
Query result:
Dictionary[386788] = 5000050
Dictionary[386789] = 5002070
Dictionary[386790] = 5003618
Dictionary[386791] = 5003629
Dictionary[386792] = 5005054
Dictionary[386793] = 5005500
Dictionary[386794] = 5007054
Dictionary[386795] = 5007878
Dictionary[386796] = 5008843
Dictionary[386797] = 5008907
Dictionary[386798] = 5008938
Dictionary[386799] = 5011358
Dictionary[386800] = 5012076
Dictionary[386801] = 5012676
Dictionary[386802] = 5013514
Dictionary[386803] = 5014629
Dictionary[386804] = 5015802

Dictionary[386805] = 5017208
Dictionary[386806] = 5017254
Dictionary[386807] = 5018781
Memory consumption after perform ‘V > 5,000,000’: 144864 kb
Compare memory consumption between lossless and lossy:
Dictionary size of ts in lossless : 387008 with size: 1548040 bytes
Dictionary size of ts in lossy : 80407 with size: 643264 bytes
As you can see, when performing lossy compression, the amount of memory and the size of dictionary reduce nearly 4 times compared to the regular dictionary.
Expression 2 has been modified to ‘100e14 < TS < 121e14’ because in sample-game.csv, the timestamp data is from 100e14.
Total row of 100e14 < TS < 121e14: 1048576
Running time of 100e14 < TS < 121e14: 0.023998s
Query results:
Dictionary[0] = 10629342490369879
Dictionary[1] = 10629347605342274
Dictionary[2] = 10629352720314738
Dictionary[3] = 10629357835286658
Dictionary[4] = 10629362950259427
Dictionary[5] = 10629368065231435
Dictionary[6] = 10629373180203890
Dictionary[7] = 10629378295176356
Dictionary[8] = 10629383410148822
Dictionary[9] = 10632024809835772
Dictionary[10] = 10632029737813340
Dictionary[11] = 10633811911716270
Dictionary[12] = 10634726820637202
Dictionary[13] = 10634731737624712
Dictionary[14] = 10634732514566388
Dictionary[15] = 10634735140223609
Dictionary[16] = 10634736654612317
Dictionary[17] = 10634737426529044
Dictionary[18] = 10634738740592737
Dictionary[19] = 10634740108248464
Memory consumption after perform 100e14 < TS < 121e14: 145964 kb
Running in lossy dictionary:
Total row of 100e14 < TS < 121e14 in lossy dict: 80407
Running time of 100e14 < TS < 121e14 in lossy dict: 0.001725s
The running time in lossy dictionary is lower than regular dictionary 0.001725s vs 0.023998s
Dictionary[0] = 10629342
Dictionary[1] = 10629347
Dictionary[2] = 10629352
Dictionary[3] = 10629357
Dictionary[4] = 10629362
Dictionary[5] = 10629368
Dictionary[6] = 10629373
Dictionary[7] = 10629378
Dictionary[8] = 10629383

Dictionary[9] = 10632024
Dictionary[10] = 10632029
Dictionary[11] = 10633811
Dictionary[12] = 10634726
Dictionary[13] = 10634731
Dictionary[14] = 10634732
Dictionary[15] = 10634735
Dictionary[16] = 10634736
Dictionary[17] = 10634737
Dictionary[18] = 10634738
Dictionary[19] = 10634740
Memory consumption after perform 100e14 < TS < 121e14 in lossy dict: 145964 kb

Description of lossy algorithm in timestamp column:
Timestamp value is accurate to picosecond (1 pico s = 10^{-12} s). We can overlook the accuracy of it by eliminating the second half of this string and the remaining part can still represent the importance of information. Its accuracy is now up to the millisecond. We can also manipulate the timestamp string by different algorithms to get different results.
For example: 10629342|490369879 -> 10629342
Pseudo code:

```
long to_lossy(string temp_col){
	return stol(temp_col.substr(0, temp_col.length() / 2));
}
```