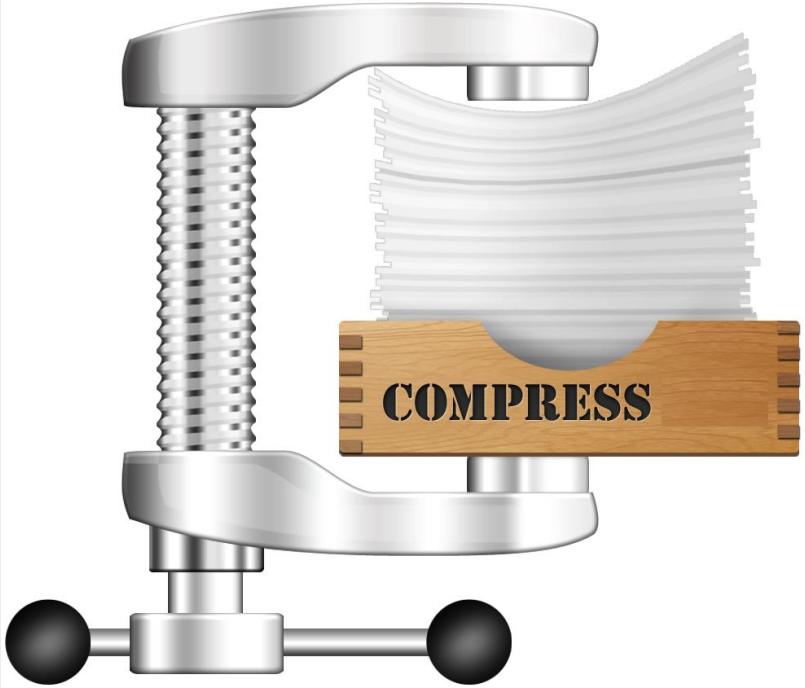
Database Compression

Tao Dai (tdai1)
Siyuan Sheng (ssheng)
Bohan Zhang (bohanz2)



Outline

- Project Objective
- Proposed Goals
- Implementation
- Evaluation
- Future Work

Project Objective

1. Use less space to store cold data
2. Reduce I/O overhead
3. Process less data per query

Dictionary-based

Compression

Dictionary-based Compression

Nation (VARCHAR 25)
China
United States
United States
China
United States

Nation (INT)
0
1
1
0
1

Dictionary:

Value(“China”)	0
Value(“United States”)	1

75% Goal (DONE)

- √ Implement dictionary-based compression for VARCHAR and VARBINARY

100% Goal (DONE)

- √ Support queries on the compressed data indirectly

  (i.e. decompress the data independently by decoding entire tile)

125% Goal (ALMOST DONE)

- √ Support queries on the compressed data directly.

  (i.e. decompress only the data needed)
- X Support join on the compressed data directly.

Implementation

- A new class *DictEncodedTile* derived from *Tile* class
- Each tile has a dictionary (original value -> encoded value)
    - original value type: VARCHAR , VARBINARY
    - encoded value type: INTEGER
    - dictionary: Hashmap

Implementation

0	Value(“CHINA”)
1	Value(“USA”)

- Varlen_val_ptrs is an array of pointers, it stores pointers that points to values.

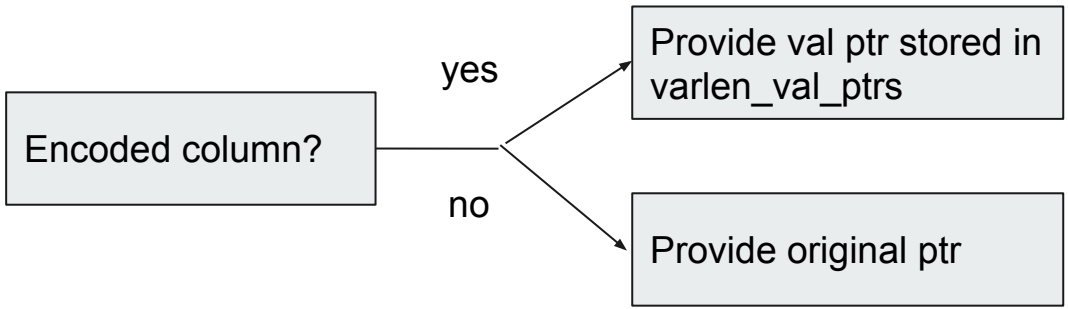
Varlen_val_ptrs:

[val0 ptr][val1 ptr] … [valn ptr]

Idx: 0 1 n

Implementation

- To query on encoded data, need to interpret index to the pointer to varlen value. Just use the stored index to do pointer arithmetic.



Val_ptr = Varlen_val_ptrs + index * 32

Correctness Test

Basic Test:

Insert data-> Compress -> Decompress :

compare the decompressed data with the original data

Select Test:

Insert data-> Compress -> Select on the compressed data:

compare the query results with the correct ones

Evaluation: Dataset

Size: 1.5 million tuples 321 MB

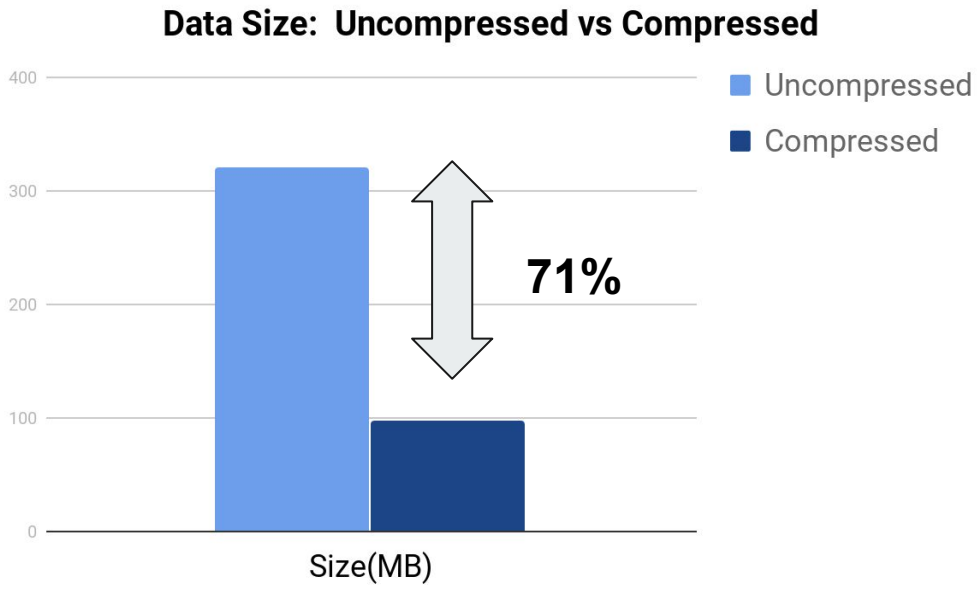
Derived from TPC-H Benchmark: Modified customer table

Source: https://github.com/electrum/tpch-dbgen

```
customer (      c_custkey     INTEGER NOT NULL,
                c_name        VARCHAR(25) NOT NULL,
                c_address     VARCHAR(40) NOT NULL,
                c_nation      VARCHAR(25) NOT NULL,
                n_comment     VARCHAR(152),
                c_region      VARCHAR(25) NOT NULL,
                r_comment     VARCHAR(152)
          );
```

Evaluation: Compression Ratio

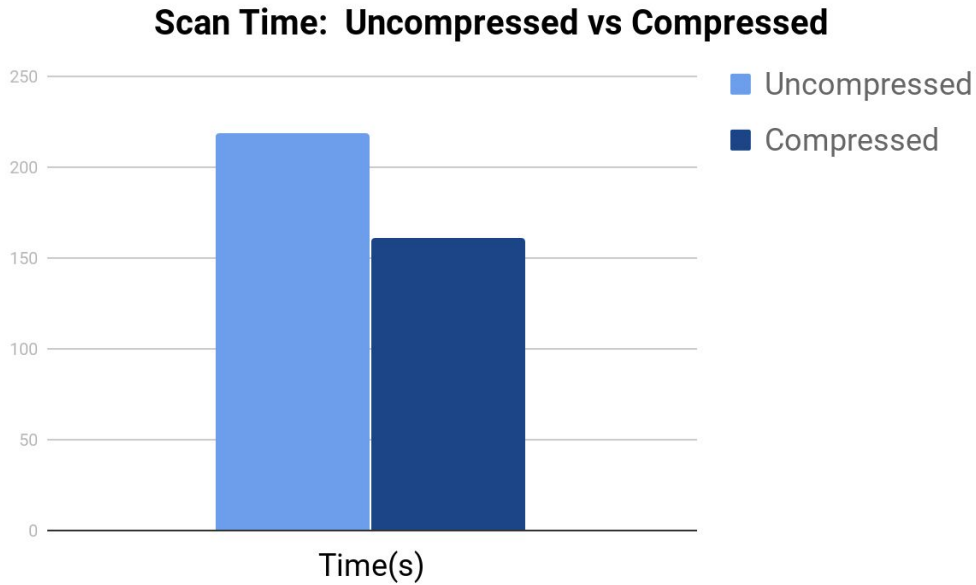
Data	Size
Uncompressed Storage	321MB
Compressed Storage	95 MB



Evaluation: Scan Timing

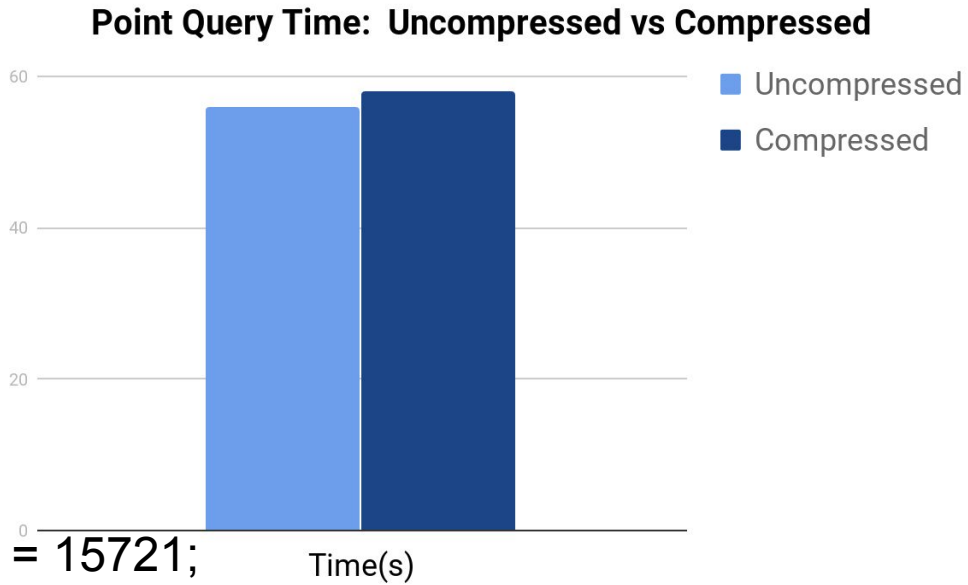
Data	Time
Uncompressed Storage	219s
Compressed Storage	161s

select * from customer;
5 times



Evaluation: Point Query Timing

Data	Time
Uncompressed Storage	56s
Compressed Storage	58s



select * from customer where c_custkey = 15721;
5 times

Modified Files

Add files: dict_encoded_tile / test

- src/storage/dict_encoded_tile.cpp
- src/include/storage/dict_encoded_tile.h
- test/storage/tile_compression_test.cpp
- test/storage/tile_compression_select_test.cpp

Modify files: codegen + tile/tilegroup

- src/codegen/proxy/runtime_functions_proxy.cpp
- src/include/codegen/runtime_functions.h
- src/codegen/runtime_functions.cpp
- src/include/codegen/tile_group.h
- src/codegen/tile_group.cpp
- src/storage/tile.cpp
- src/storage/tile_group.cpp
- src/include/storage/tile_group.h
- src/include/storage/tile.h

Conclusion

- Significantly reduce data size for frequent long strings (varchar)
- Achieve equivalent or even better query performance compared to uncompressed data

Future Work

- Support join on the compressed data
- Implement order-preserving compression for range queries

Thank you.