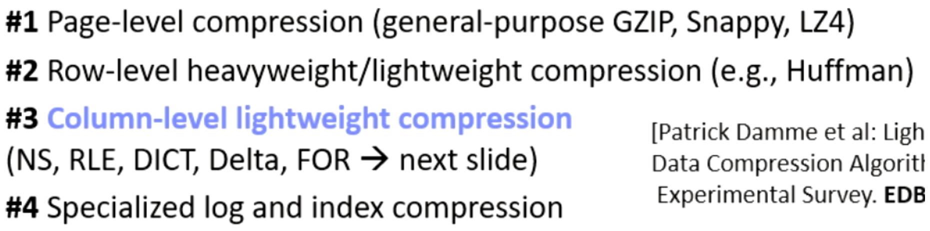
DB Compression Overview

- fit larger datasets in memory
- less I/O
- better cache uitilization
- some DBs allow query processing directly on compressed data
  - 

    **#1** Page-level compression (general-purpose GZIP, Snappy, LZ4)

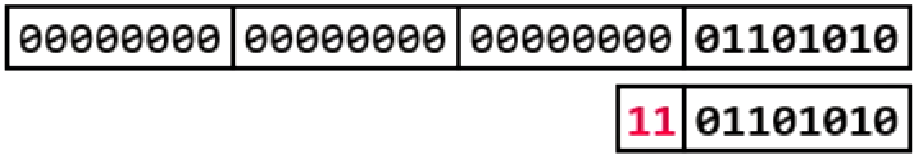
    **#2** Row-level heavyweight/lightweight compression (e.g., Huffman)

    **#3** Column-level lightweight compression (NS, RLE, DICT, Delta, FOR → next slide)

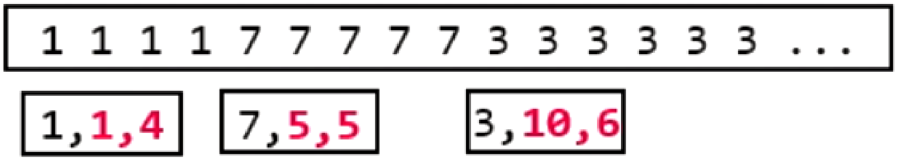
    **#4** Specialized log and index compression

    [Patrick Damme et al: Ligh
    Data Compression Algorith
    Experimental Survey. **EDB**

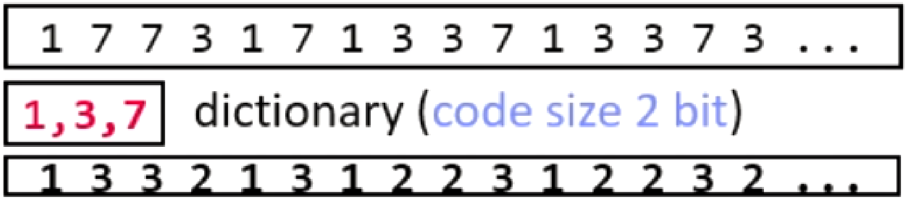
Lightweight Database Compression Schemes

- null suppresion
  - compress integers with leading zeros
  - 

    00000000 00000000 00000000 01101010

    11 01101010
- run-length encoding
  - compress sequences of equal values by “runs”
  - each run consists of
    - value
    - start
    - length
  - 

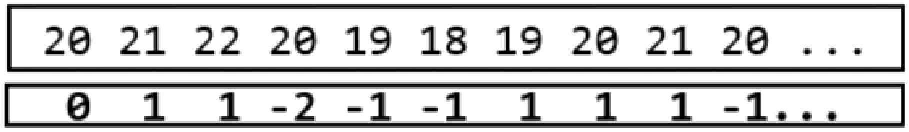
    1 1 1 1 7 7 7 7 7 3 3 3 3 3 3 ...

    1,1,4 7,5,5 3,10,6
- dictionary encoding
  - compress column with few distinct values
  - create dictionary with all values
  - store pos in dictionary instead of actual value
  - 

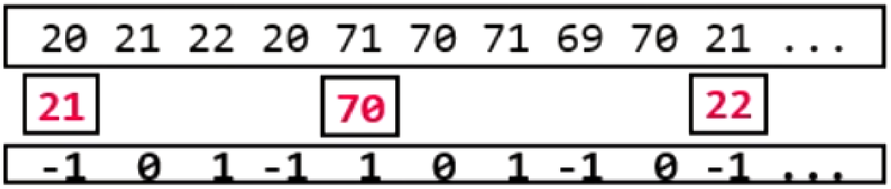
    1 7 7 3 1 7 1 3 3 7 1 3 3 7 3 ...

    1,3,7 dictionary (code size 2 bit)

    1 3 3 2 1 3 1 2 2 3 1 2 2 3 2 ...
    - compression would be more effective if values were strings instead

- delta encoding
  - compress sequence with small changes
  - store delta/change to previous value
  - 

    20 21 22 20 19 18 19 20 21 20 ...

    0 1 1 -2 -1 -1 1 1 1 -1...
- frame-of-reference encoding
  - compress values by storing delta to reference value
  - outlier handling
  - 

    20 21 22 20 71 70 71 69 70 21 ...

    21 70 22

    -1 0 1 -1 1 0 1 -1 0 -1 ...

[[Physical Design]]