Lec 4 DB Storage II

Recall:
- Disk oriented architecture
- Page-oriented scheme
- Slotted page organisation

Insertion:
1. Check page directory for page w free space
2. Retrieve page from disk, if not cached
3. Check slot array to find space

Update:
1. Check page directory for page num
2. Retrieve page from disk
3. Find offset in slot array
4. If enough space for new data:
   overwrite data
   Else
   mark as deleted and reinsert

Problems with tuple-oriented

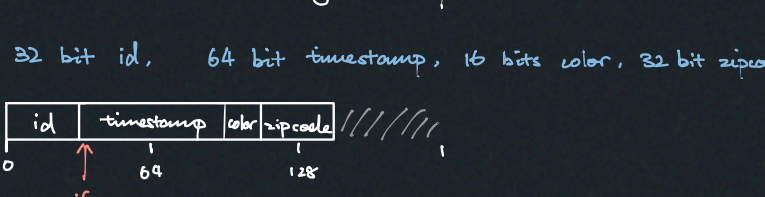
- Fragmentation — empty space available but not contiguous
  - Need clean up mechanism
  - .. forwarding ..
- Useless Disk IO — need to read entire page to update one thing
  - Sorting records on disk could help
  - Optimal page size depends on usage
- Random Disk IO — slow IO if lacking physical locality
  - Also hardware dependent

Log-Structured Storage

* B-tree
- similar to Tree but balanced & each node is page-size
- O(log n) search, with huge base
- key-val pairs
- data at leaf
- fast read, slow write

* LSFS (Log-Structured File Sys)
- data throughout tree
- level 0 stay in mem, others on disk
- updates kept in log until
- fast write, slow read
- each node is sorted str table (SSST)
- lower levels bigger usually

level 0 — "mem table"



- if mem table full: lock it, flush level 0 to level 1, sort table 1, make new mem table for level 0
  ("Levelling method" as in RocksDB)
  recur to compact lower levels as needed
- summary table: min / max of each SST, bloom filter] allow skipping for better performance
- If looking for non-existing key: has to go down all levels

The log:
- Put every PUT and DEL on log ← defer to compaction step

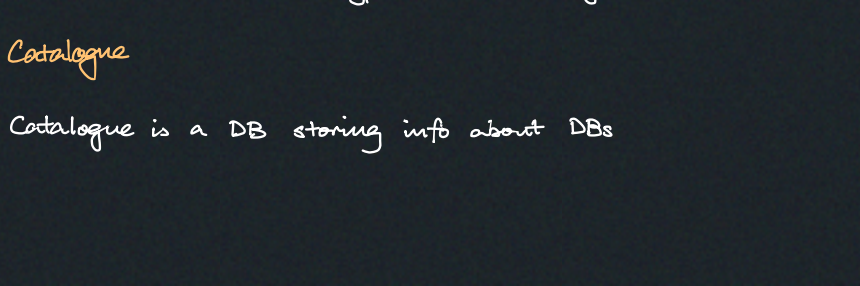
Compaction: → Ex. on 「1」
- Merge two SST into a larger SST
- Like tracing the log to reconstruct latest values

* Compaction Algorithm

Log-Structured Merge (LSM) considerations:
read/write/space amplification tradeoff

Index-Oriented Storage

B-Tree, with data at leaf



Tuple Storage — Data Layout

Ex. (id int primary key, value bigint)

char[]

Non-word size aligned tuples

32 bit id, 64 bit timestamp, 16 bits color, 32 bit zipcode

maybe pad + pack

maybe rearrange

Data Repr

int / bigint / smallint / tinyint — match C++ types

float / real — can round badly i.e. off by a cent

numeric / decimal — more precise, available in most SQL impl, software level impl, not hardware float, much bigger & slower

varchar ...

null — per column bit map; some special value repr's null (e.g. INT32_MIN); per attribute null flag (viz. one bit for each field)

Large values
- put inside an overflow page, and point to it
- if overflowing overflow page → chain them
- blob type: put in file sys, and refer to it

Catalogue

Catalogue is a DB storing info about DBs