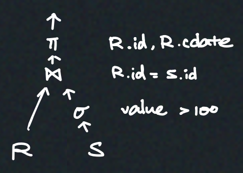
Lec 10 Sorting and Aggregation

Preview

Query plan



Algorithm
- Maximise sequential IO
- Spill to disk if necessary
- Sort when ORDER BY
  ... DISTINCT, GROUP BY

In - memory Sorting

→ Just run sorting algorithm
  └ Quicksort, TimSort, Insertion Sort
    └ hybrid merge sort & insertion sort

  └ Perf depends on data distribution
    └ nearly sorted — simple ones may be faster

Top-n heap sort

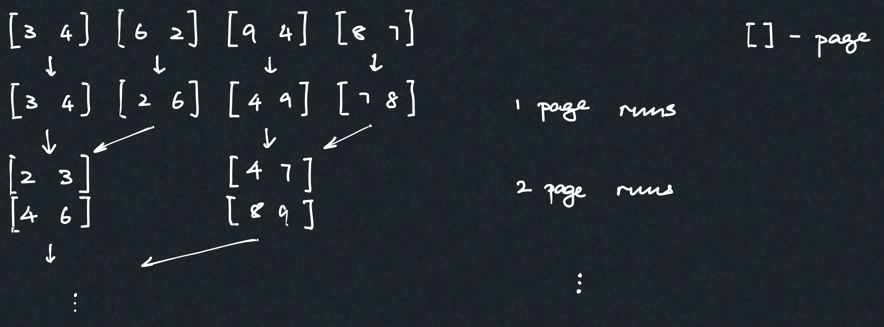
If only want top-n elems, iteratively put things in sorted size-n heap (discard things when appropriate when heap is full)
viz. keep top-n so far while scanning input

External merge sort — sort things larger then mem

D&C into multiple smaller runs, sort them, then combine
→ Keep keys & point to data, sort, then chase pointer
→ Keep keys & data, then sort (early materialisation)

▷ 2 way external merge sort

Sort each run, write to new tmp file, keep merging



Whole algorithm	need 3 pages in buf pool
Num passes	$1 + \lceil \log_2 N \rceil$
IO cost	$2N \cdot$ num passes

Generalise
Optimise
- increase fanout, multi-way merge
  └ use heap to find min
- go straight to higher page run
  ↙ step func!

⇒ Num passes $1 + \lceil \log_{B-1} \lceil \frac{N}{B} \rceil \rceil$ B = buffer size
IO cost $2N \cdot$ num passes

- Double buffering : prefetch data for next run in background while merging. Requires double buff pool size — should use if halving B doesn't bump num passes

- Code opt. : inline the comparison func

- String sorting : try use prefix

B-Tree Sorting
└ If clustered (leaves in physical order), then good idea to run sorting on those pages. Else bad idea

Aggregation

▷ DISTINCT

→ filter, get column, sort, scan & remove duplicate
  └ still sorted after dedup
→ external hashing
  1. Partition — two indep hash funcs h_1, h_2
     filter, get column, partition by h_1 into buckets
  2. Rehashing (hopefully hash table fit in memory)
     rehash each bucket into hash table via h_2 (on RAM or with disk hash table)

▷ GROUP BY, calc AVG

→ keep running aggregation in mem as hash table.
  └ hopefully in mem
  └ for AVG, keep running sum & count
    ⋮ for COUNT, MIN, MAX, SUM, etc.