COS 597A:
Principles of
Database and Information Systems

File organization and access costs

Move down a level of abstraction

- Until now at level of user view of data
  - models
  - query languages
- Now: how actually store data and access
  - disk storage (low-level abstraction)
  - file organization (level between disk and user)
  - access costs
- Next: how compute query results efficiently
  - what are algorithms
  - what are costs

Disks

- Main storage for large databases
  - too much data for main memory
  - need permanent storage

So far as technology advances, disk (aka hard drive) still gives significantly more space and less speed, regardless of how big/cheap RAM gets
  - voracious appetite for space!
  - True no matter where sit on cost/size curve for system
- impact solid state drives?

Disk organization

- platters containing tracks
- track read sequentially
- can seek from track to track
- tracks broken into sectors
  - smallest physical unit can read / address
  - typical size 512 Bytes
    - Advanced Format 4096 Bytes

Disk access costs

- seek time
  - milliseconds
- rotational latency
  - milliseconds
- transfer rate
  - 100 MB/sec
- compare RAM
  - nanoseconds
  - factor of 10^6

- disk closeness
  - adjacent sectors
  - same track
  - same cylinder
  - adjacent cylinder

File

- collection of records
- records grouped into pages
  - record ID (rid) conceptually (page #, slot #)
  - Slot # gives position on page
- page is multiple of disk sectors
  - stored sequentially on disk
  - page smallest unit read
    - typical 4-8 KB
  - "page" also known as "block"
  -

Memory buffer

- Memory allocated for file read/write (I/O)
- size of buffer in pages
- read disk page into memory buffer
- write to disk page from memory
- buffer as big as can afford
- buffer often not big enough
  - buffer management

File organizations

Two issues

- how records assigned pages
  - affects algorithms
  - affects which pages read & in what order
- how pages put on disk
  - want pages of file physically close on disk
  - want likely sequences of pages read close

File storage management

- Who manages storage of files on disk
  1. custom OS for DBMS
  2. let OS do it
     - typically one file per relation
  3. define one OS file for whole DBMS
     - DBMS manages w/in file
- DBMS buffer manager
  - replacement strategy
  - pinning
  - forced-out pages

Conceptual organization of file

- Heap file
  - linked list pages or directory of pages
  - no order records in pages
  - pages anywhere on disk

Conceptual organization of file (cont.)

- Hashing file
  - hash function applied to record puts in bucket
    - gives address of primary page of bucket
    - designated hash attribute(s) of records
  - pages can be anywhere if hash gives location
  - can be overflow
    - pointers to overflow pages
    - where overflow pages on disk?
  - try to keep pages 80% full

Conceptual organization of file (cont.)

- Sequential file
  - conceptually ordered set of records
    - order often sort on attributes of relation
  - records stored in order giving ordered set pages
  - pages sequentially close => physically close
    - compact after delete
  - binary search?
    - need i^{th} page in sorted order in one disk I/O
- can have sorted file that is not sequential file

Acces cost model

- B number of data pages in file
- R number of records per page in full page
- D average time to R/W disk page
  - assume individual pages not sequential on disk
    - no "block reads"
- Ignore CPU time

Simple average case time analysis

- Simple assumptions
  - Insert at end of heap
  - No overflow buckets for hash
    - Keep 80% occupancy
    - Inserts/deletes in balance
  - Sorted sequential file with binary search
  - Delete assumes have address of record
- Use analysis for relative costs
  - TOO CRUDE for "on the fly" cost estimates

B data pages in file
R records per page

D avg time to R/W page

Search on record attribute

Avg. time	Heap	Sorted	Hashed
Scan			
Search = (unique)			
Search = (multiple)			
Search range			
Insert			
Delete			

Avg. time	Heap	Sorted	Hashed
Scan	BD	BD	1.25 BD
Search = (unique)	.5BD	$D\log_2 B$	D
Search = (multiple)	BD	**$D(\log_2 B$ + # extra matching pages)**	**D (1 + # extra matching pages)**
Search range	BD	"	1.25 BD
Insert	2D	**Search + D + BD**	2D
Delete	2D	2D+BD	2D

Critique

- R&G don't account for how to keep hashed file 80% occupied
  - if not, overflow costs sometimes
- Sorted sequential file - expensive to keep pages continguous on disk
  - link pages + look-up table sorted on first value on page of attribute sorted on

file page #	file page location	first attribute value of page

=> indexes