COS 597D:
Principles of
Database and Information Systems

Storage Organization and Data Access

1

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

2

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 (SSDs)?

3

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

4

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

5

Data 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”

6

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

7

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

8

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

9

Conceptual organization of file

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

10

Conceptual organization of file (cont.)

- Hashing file
  – hash function puts record in bucket
    - bucket size is some number of pages
    - hash 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

11

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

12

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

13

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

14

B data pages in file
R records per page
D avg time to R/W page

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 (have record location)	2D	2D+**BD**	2D

15

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

  => index
- Improvements only for attribute of sort or hash
  - Improve access using other attributes?

    => index

16

Index

- Auxillary information on location of a record or page to facilitate retrieval
- Search key: attribute (i.e. field, column) used as look-up value for index
  - not confuse with {primary, candidate, super} key
  - alternate term "index field"
    - "index key" if attribute is a candidate key
  - Could actually be combination of attributes
    - e.g. LastName, FirstName
- Basic index is a file containing mappings:

  Seach key value ⟶ pointer(s) to page(s) containing records with given search key value

17

Index Types

1. Index works *with* file organization
   - Index and file work off same attribute
   - Two types:
     - A. Index **is** file organization
       - Example: Hashing file organization
       - Index is access method: get pointer to page serving as primary bucket of records for given hash value
     - B. Index supplements file organization
       - Example: Sequential file plus search tree whose leaves point to first page containing value seeking
   - called clustered index
   - some refer to as primary index
     - not necessarily on primary key of relation

18

Index Types cont.

2. Index works independent of file organization
   - File not organized on search key of index
   - Index must provide
     search key value → list of pointers to
     all file pages that contain
     records with that value
   - Example hash index:
     - bucket contains list of page pointers
     - pages may be scattered throughout the file
     - overflow if too many pointers for one bucket
   - called nonclustering index
   - some refer to as secondary index

19

A Sorted Index

- Consider sorted file but without consecutive pages stored adjacently on disk
  - Each page sorted
  - Each page linked to next page in sorted order
  - ***Cannot*** binary search
- Index:

sorted attribute value	pointer to first page containing
↑ Sorted order ↓	

- One entry per attribute value in data file => dense index
- Can binary search index entries if can keep in memory or in sequential disk pages

20

Alternative sparse index for sorted file

again:
index search key same as sort attribute for file

file page number	page location	first value of search key on page
		↑ Sorted order ↓

One entry *per file page*
Again, binary search if keep in memory or sequentially on disk

21

Cost example dense sorted index

- Use our crude estimates with
  B data pages in file **D** avg time to R/W page
  R records per page
- Suppose index record 1/10 size of data record
- Suppose search key (= sort attribute) is candidate key
- Cost search for unique value using dense index:

  number of records is the same for index file
  B/10 pages in index file (file page size is fixed for all files)
  Binary search cost = $D\log_2(B/10)$

  Total cost = $D\log_2(B/10) + D$
  includes data page access

22

Cost example sparse sorted index

- Use our crude estimates with
  B data pages in file **D** avg time to R/W page
  R records per page
- Suppose index record 1/10 size of data record
- Suppose search key (= sort attribute) is candidate key
- Cost search for unique value using sparse index:

  B pages in data file => B entries in index file
  10R index records per file page => B/(10R) index pages
  Binary search cost = $D\log_2(B/(10R))$

  Total cost = $D\log_2(B/(10R)) + D$
  includes data page access

23

Compare costs:

- Use our crude estimates with
  B data pages in file **D** avg time to R/W page
  R records per page
- Suppose index record 1/10 size of data record
- Suppose search key (= sort attribute) is candidate key
- Cost search for unique value using dense index?
  $D\log_2(B/10) + D$
- Cost search for unique value using sparse index?
  $D\log_2(B/(10R)) + D$

24

Compare costs: insertion

- Use our crude estimates with
  - **B** data pages in file **D** avg time to R/W page
  - **R** records per page
- Suppose index record 1/10 size of data record
- Suppose search key (= sort attribute) is candidate key
- Recall data file pages not nec. stored consecutively on disk
  - so can use overflow pages
- Cost to insert = cost to insert in data file
  + cost to insert in index file

  = Search cost
  + D + ~4D write data file page and move ~1/2 records of page if overflow
  + D write index entry
  + { D*B/10 move records for dense index ; D*B/(10R) move records for sparse index }

Index independent of file organization

But look again,
if search key is a *candidate key*,
this index works for *any* file organization :

search key	pointer to *unique* page containing
↑ Sorted order ↓	

One entry per search key value - dense
Can binary search index as before if keep in memory or sequentially on disk

Sorted index for general case

- One value of search key found in many records
- Need list of pointers to pages containing these records
- Dense index still works
- Most common arrangement:
  - indirection

Seach key	pointer to page containing list
↑ One entry per attribute value. Sorted order ↓	

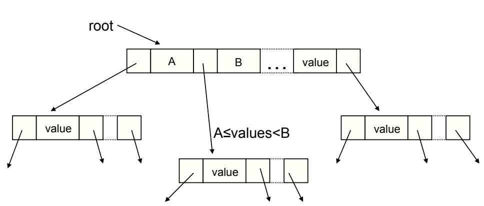
Addressing costs

- Large sorted index costly in space and in time to insert/delete
  - When sorted index clustered, can use sparse index to avoid space
  - For general case, *must* have dense index
- Ideal: index to fit on one file page.
  - Keep in main memory
- Rarely achieve, so next best:
  - Index need not be stored sequentially on disk
  - Access cost is no worse than $O(\log_2 B)$

=> **Search Tree!**

Tree index

- Each node of tree fits in one page
- Each node of tree contains search key values and pointers to subtrees for ranges of values
- A leaf is
  - For clustered index: a page of data file
  - For general index: a page of pointers to records with given index values



Static Trees

- Build for file of records as balanced tree
- Not gracefully accommodate insert/delete
- ISAM: Indexed Sequential Access Method
- We focus on dynamic search trees