Storage Structures Introduction
Chapter 8 (3rd edition)

Sharma Chakravarthy
UT Arlington
sharma@cse.uta.edu

Disks and Files

☞DBMS stores information on ("hard") disks.
☞This has major implications for DBMS design! Remember impedance mismatch!

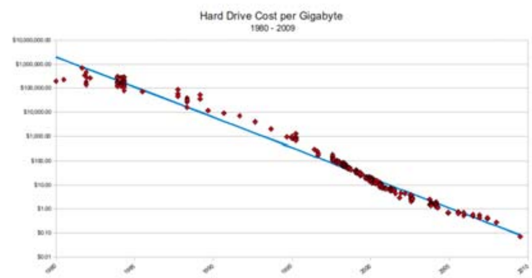
- READ: transfer data from disk to main memory (RAM).
- WRITE: transfer data from RAM to disk.

- Both are high-cost operations, relative to in-memory operations, so must be planned carefully!

Why Not Store Everything in Main Memory?

☞*Costs too much*. $80 will buy you either 8 GB (4 GB 2 year ago) of RAM or 1 to 1.5 TB (250 GB 1 year ago) of disk today or 64GB of SSD (was $250 3 years ago)
☞*Main memory is volatile*. We want data to be saved (persistent) between runs. (Obviously!)
☞Typical storage hierarchy:
- Cache – most expensive
- Main memory (RAM) for currently used data.
- Solid state disks (SSD) -- New !! (up to 512 GB)
- Disk for the main database (secondary storage).
- Other storage devices (flash card, usb stick, …)
- Tapes for archiving older versions of the data (tertiary storage).
- DVD/CDROM and other devices – least expensive

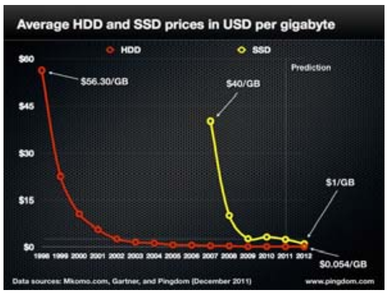
Over the last 30 years, space per unit cost has doubled roughly every 14 months (increasing by an order of magnitude every 48 months)



SSD Vs. HDD

☞HDD 3c/gb in 2014

☞SSD is .35c/gb in 2014



☞From: www.tomshardware.com

Disks

☞Secondary storage device of choice.

☞Main advantage over tapes: <u>*random access*</u> vs. *sequential*.

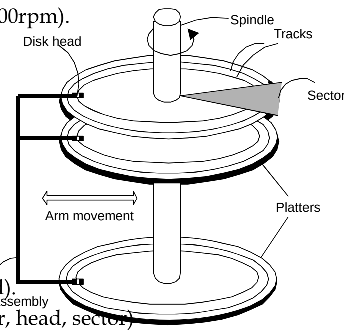
☞Data is stored and retrieved in units called *disk blocks* or *pages.*

☞Unlike RAM, time to retrieve a disk page varies depending upon location on disk.

- Therefore, relative placement of pages on disk has major impact on DBMS performance!

Components of a Disk

- The platters spin (say, 7200rpm).
- The arm assembly is moved in or out to position a head on a desired track. Tracks under heads make a *cylinder* (imaginary!).
  - Only one head reads/writes at any one time.
- *Block size* is a multiple of *sector size* (which is fixed).
- Addressing: CHS (cylinder, head, sector)



Accessing a Disk Page

☞Time to access (read/write) a disk block:

- *seek time* (moving arms to position disk head on track)
- *rotational delay* (waiting for block to rotate under head)
- *transfer time* (actually moving data to/from disk surface)
- Buffer size (2 MB typical, 8 MB, …)

☞Seek time and rotational delay dominate.

- Seek time varies from about 1 to 20msec
- Rotational delay varies from 0 to 10msec
- Transfer rate is about 1 msec per 4KB page

☞Key to lower I/O cost: reduce seek/rotation delays! Hardware vs. software solutions?

Accessing a Disk Page

☞Time to access (read/write) a disk block:
- *Average seek time -- 9.1 msec*
- *Average rotational delay -- 4.17 msec*
- *transfer rate – 13MB/sec*
- Seek from one track to next – 2.2 msec
- Max. seek time 15 msec

☞Disk access takes about 10 msec whereas accessing memory location takes about 60 nano secs !!

☞Memory is more than a Million times faster!!

Slide 9

Arranging Pages on Disk

☞`*Next*' block concept:
- blocks on same track, followed by
- blocks on same cylinder, followed by
- blocks on adjacent cylinder

☞Blocks in a file should be arranged sequentially on disk (by `next'), to minimize seek and rotational delay.

☞For a sequential scan, *pre-fetching* several pages at a time is a big win!

Slide 10

Disk Space Management

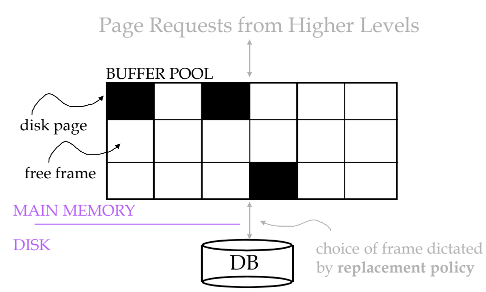
☞Lowest layer of DBMS software manages space on disk.

☞Higher levels call upon this layer to:
- allocate/de-allocate a page
- read/write a page

☞Request for a *sequence* of pages must be satisfied by allocating the pages sequentially on disk! Higher levels don't need to know how this is done, or how free space is managed.

Slide 11

Buffer Management in a DBMS



☞*Data must be in RAM for DBMS to operate on it!*

☞*Mapping of <frame#, pageid> pairs is maintained.*

Slide 12

When a Page is Requested ...

- If requested page is not in pool:
  - Choose a frame for *replacement*
  - If frame is dirty, write it to disk
  - Read requested page into chosen frame
- *Pin (increment pin_count)* the page and return its address.
- If there is no frame to choose, then the buffer is full. *When is the buffer full*?

* *If requests can be predicted (e.g., sequential scans) pages can be pre-fetched several pages at a time!*

Slide 13

More on Buffer Management

- Requestor of page must unpin it, and indicate whether page has been modified:
  - *dirty* bit is used for this.
- Page in pool may be requested many times,
  - a *pin count* is used. A page is a candidate for replacement iff *pin count* = 0.
- CC & recovery may entail additional I/O when a frame is chosen for replacement. (*Write-Ahead Log* protocol; more later.)

Slide 14

Buffer Replacement Policy

- Frame is chosen for replacement by a *replacement policy:*
  - Least-recently-used (LRU), Clock, most-recently-used (MRU) etc.
- Policy can have big impact on # of I/O's; depends on the *access pattern*.
- *Sequential flooding*: Nasty situation caused by LRU + repeated sequential scans.
  - # buffer frames is less than # pages in file means each page request causes an actual I/O. MRU much better in this situation (but not in all situations, of course).

Slide 15

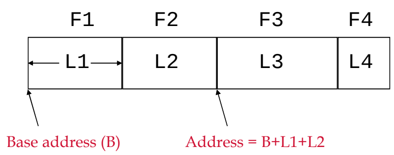
DBMS vs. OS File System

OS does disk space & buffer management
why not let OS manage these tasks?

- Differences in OS support: portability issues
- Some limitations, e.g., files can't span disks.
- Buffer management in DBMS requires ability to:
  - pin a page in buffer pool, force a page to disk (important for implementing CC & recovery),
  - adjust *replacement policy,* and pre-fetch pages based on access patterns in typical DB operations.

Slide 16

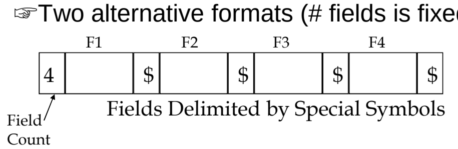
Record Formats: Fixed Length



- Information about field types same for all records in a file; stored in *system catalogs.*

Record Formats: Variable Length

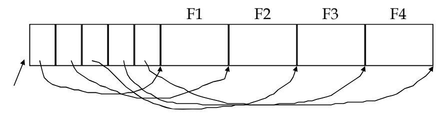
- Two alternative formats (# fields is fixed):



- Finding *i'th* field requires scan of record

Record Formats: Variable Length

- Two alternative formats (# fields is fixed):



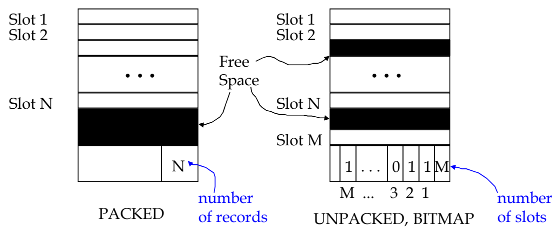
Array of Field Offsets

* Second offers direct access to i'th field, efficient storage of nulls (special *don't know* value); small directory overhead.

Subtle Issues: Variable Length

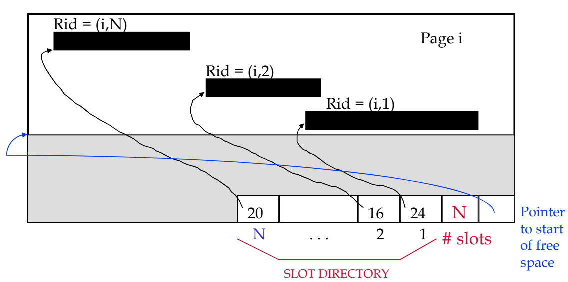
- What about modification? Growth may involve moving fields
- What if a record does not fit into the space remaining on a page? (rids and forwarding address)
- A record grows to occupy more than one page! (chained smaller records) – spanning records

Page Formats: Fixed Length Records (physically numbered slots)



* Record id (rid) or tuple id (tid) = <page id, slot #>. *In the first alternative, moving records for free space management changes rid; may not be acceptable.*

Page Formats: Variable Length Records (logically numbered slots)



* *Can move records on page without changing rid; so, attractive for fixed-length records too.*
* Record cannot be removed from the directory! Why?
* Can be compacted without changing rid

Files of Records

☞Page or block is OK when doing I/O, but higher levels of DBMS operate on *records*, and *files of records*.

☞FILE: A collection of pages, each containing a collection of records. Must support:

- insert/delete/modify opeartions
- read a particular record (specified using *record id*)
- scan all records (possibly with some conditions on the records to be retrieved)

Unordered (Heap) Files

☞Simplest file structure contains records in no particular order.

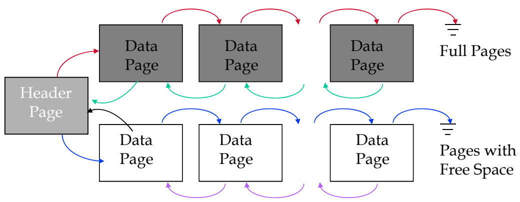
☞As a file grows and shrinks, disk pages are allocated and de-allocated.

☞To support record level operations, we must:

- keep track of the *pages* in a file
- keep track of *free space* on pages
- keep track of the *records* on a page

☞There are many alternatives for keeping track of this.

Heap File Implemented as a List



- ☞The header page id and Heap file name must be stored someplace.
- ☞Each page contains 2 `disk pointers' plus data.
- ☞Disadvantage for variable length records! (all pages will be on the free list)

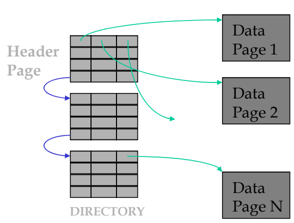
Slide 25

Indexes

- ☞A Heap file allows us to retrieve records:
  - by specifying the *rid*, or
  - by scanning all records sequentially
- ☞Sometimes, we want to retrieve records by specifying the *values in one or more fields*, e.g.,
  - Find all students in the "CS" department
  - Find all students with gpa > 3.0
- ☞Indexes are file structures that enable us to answer such value-based queries efficiently.

Slide 27

Heap File Using a Page Directory



- ☞The entry for a page can include the number of free bytes on the page.
- ☞The directory is a collection of pages; linked list implementation is just one alternative.
  - *Much smaller than linked list of all HF pages!*

Slide 26

System Catalogs

- ☞For each index:
  - structure (e.g., B+ tree) and search key fields
- ☞For each relation:
  - name, file name, file structure (e.g., Heap file)
  - attribute name and type, for each attribute
  - index name, for each index
  - integrity constraints
- ☞For each view:
  - view name and definition
- ☞Plus statistics, authorization, buffer pool size, etc.

* *Catalogs are themselves*
* *stored as relations!*

Slide 28

Attr_Cat(attr_name, rel_name, type, position)

attr_name	rel_name	type	position
attr_name	Attribute_Cat	string	1
rel_name	Attribute_Cat	string	2
type	Attribute_Cat	string	3
position	Attribute_Cat	integer	4
sid	Students	string	1
name	Students	string	2
login	Students	string	3
age	Students	integer	4
gpa	Students	real	5
fid	Faculty	string	1
fname	Faculty	string	2
sal	Faculty	real	3

Summary

☞Disks provide cheap, non-volatile storage.

- Random access, but cost depends on location of page on disk; important to arrange data sequentially to minimize *seek* and *rotation* delays.

☞Buffer manager brings pages into RAM.

- Page stays in RAM until released by requestor.
- Written to disk when frame chosen for replacement (which is sometime after requestor releases the page).
- Choice of frame to replace based on *replacement policy (LRU, MRU, FIFO, CLOCK. …).*
- Tries to *pre-fetch* several pages at a time.

Summary (Contd.)

☞DBMS vs. OS File Support

- DBMS needs features not found in many OSs, e.g., forcing a page to disk, controlling the order of page writes to disk, files spanning disks, ability to control pre-fetching and page replacement policy based on predictable access patterns, etc.

☞Variable length record format with field offset directory offers support for direct access to i'th field and null values.

☞Slotted page format supports variable length records and allows records to move on page.

Summary (Contd.)

☞File layer keeps track of pages in a file, and supports abstraction of a collection of records.

- Pages with free space identified using linked list or directory structure (similar to how pages in file are kept track of).

☞Indexes support efficient retrieval of records based on the values in some fields.

☞Catalog relations store information about relations, indexes and views. (*Information that is common to all records in a given collection.*)

Thank You !