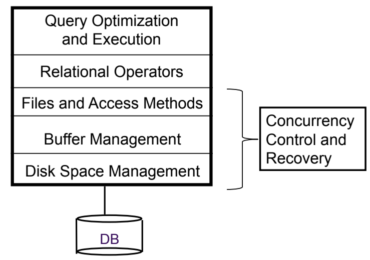
Storing Data: Disks and Files

Lecture 3

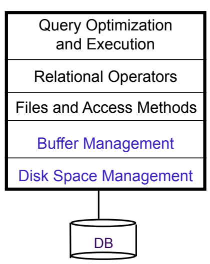
(R&G Chapter 9)

"Yea, from the table of my memory
I'll wipe away all trivial fond records."
-- Shakespeare, *Hamlet*

Block diagram of a DBMS



Disks, Memory, and Files



Disks and Files

- DBMS stores information on disks.
  - Disks are a mechanical anachronism!
- Major implications for DBMS design!
  - READ: transfer data from disk to main memory (RAM).
  - WRITE: transfer data from RAM to disk.
  - Both high-cost relative to memory references
    - Can/should plan carefully!

Why Not Store Everything in Main Memory?

- *Costs too much.* For ~$1000, PCConnection will sell you either
  - ~80GB of RAM (unrealistic)
  - ~400GB of Flash USB keys (unrealistic)
  - ~180GB of Flash solid-state disk (serious)
  - ~7.7TB of disk (serious)
- *Main memory is volatile.*
  - Want data to persist between runs. (Obviously!)

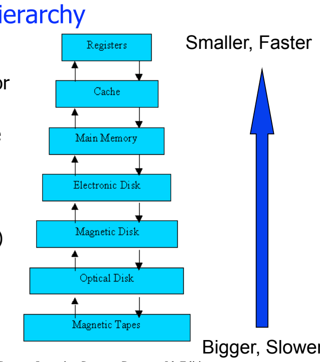
The Storage Hierarchy

–Main memory (RAM) for currently used data.

–Disk for main database (secondary storage).

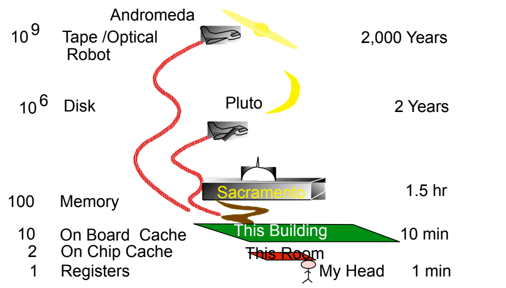
–Tapes for archive (tertiary storage).

–The role of Flash (SSD) still unclear



Source: Operating Systems Concepts 5th Edition

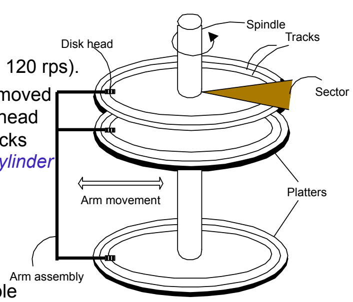
Jim Gray's Storage Latency Analogy: How Far Away is the Data?



Disks

- Still the secondary storage device of choice.
- Main advantage over tape:
  - *random access* vs. *sequential*.
- Fixed unit of transfer
  - Read/write *disk blocks* or *pages* (8K)
- *Not* "random access" (vs. RAM)
  - Time to retrieve a block depends on location
  - Relative placement of blocks on disk has major impact on DBMS performance!

Components of a Disk



The platters spin (say, 120 rps).

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).

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)
- Seek time and rotational delay dominate.
  - Seek time varies from 0 to 10msec
  - Rotational delay varies from 0 to 3msec
  - Transfer rate around .02msec per 8K block
- Key to lower I/O cost: reduce seek/rotation delays! Hardware vs. software solutions?

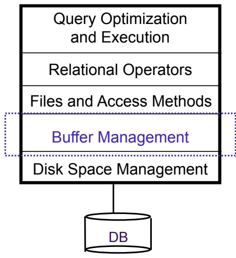
Arranging Pages on Disk

- `*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!

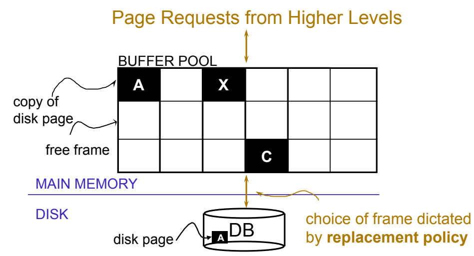
Disk Space Management

- Lowest layer of DBMS, 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 best satisfied by pages stored sequentially on disk!
  - Responsibility of disk space manager.
  - Higher levels don't know how this is done, or how free space is managed.
  - Though they may make performance assumptions!
    - Hence disk space manager should do a decent job.

Context



Buffer Management in a DBMS



- *Data must be in RAM for DBMS to operate on it!*
- *BufMgr hides the fact that not all data is in RAM*

When a Page is Requested ...

- Buffer pool information table contains:
  <frame#, pageid, pin_count, dirty>

1. If requested page is not in pool:
   a. Choose a frame for *replacement.*
      Only "un-pinned" pages are candidates!
   b. If frame "dirty", write current page to disk
   c. Read requested page into frame
2. *Pin* the page and return its address.

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

More on Buffer Management

- Requestor of page must eventually:
  1. *unpin* it
  2. indicate whether page was modified via *dirty* bit.
- Page in pool may be requested many times,
  - a *pin count* is used.
  - To pin a page: pin_count++
  - A page is a candidate for replacement iff *pin count* == 0 (*"unpinned"*)
- CC & recovery may do additional I/Os upon replacement.
  - *Write-Ahead Log* protocol; more later!

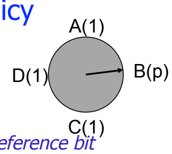
Buffer Replacement Policy

- Frame is chosen for replacement by a *replacement policy:*
  - Least-recently-used (LRU), MRU, Clock, …
- Policy can have big impact on #I/O's;
  - Depends on the *access pattern.*

LRU Replacement Policy

- *Least Recently Used (LRU)*
  - (Frame pinned: "in use", not available to replace)
  - track time each frame last *unpinned* (end of use)
  - replace the frame which has the earliest unpinned time
- Very common policy: intuitive and simple
  - Works well for repeated accesses to popular pages
- *Problem: Sequential flooding*
  - LRU + repeated sequential scans.
  - # buffer frames < # pages in file means each page request causes an I/O.
  - Idea: MRU better in this scenario? We'll see in HW1!

"Clock" Replacement Policy



- An approximation of LRU
- Arrange frames into a cycle, store one *reference bit per frame*
  - Can think of this as the 2nd chance bit
- When pin count reduces to 0, turn on ref. bit

- When replacement necessary:

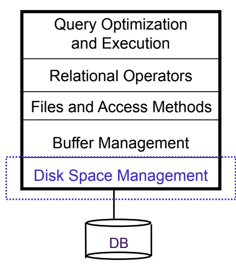
```
do for each page in cycle {
   if (pincount == 0 && ref bit is on)
         turn off ref bit; // 2nd chance
   else if (pincount == 0 && ref bit is off)
         choose this page for replacement;
} until a page is chosen;
```

DBMS vs. OS File System

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

- Buffer management in DBMS requires ability to:
  - pin a page in buffer pool, force a page to disk & order writes (important for implementing CC & recovery)
  - adjust *replacement policy,* and pre-fetch pages based on access patterns in typical DB operations.
- I/O typically done via lower-level OS interfaces
  - Avoid OS "file cache"
  - Control write timing, prefetching

Context



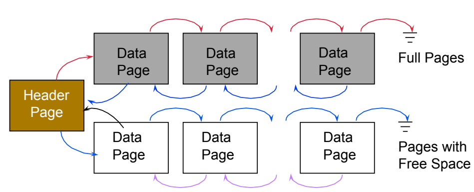
Files of Records

- Blocks are the interface for 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 record
  - fetch a particular record (specified using *record id*)
  - scan all records (possibly with some conditions on the records to be retrieved)
- Typically implemented as multiple OS "files"
  - Or "raw" disk space

Unordered (Heap) Files

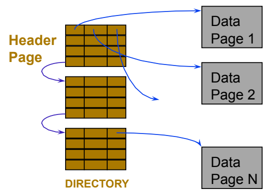
- Collection of records in no particular order.
- As file shrinks/grows, disk pages (de)allocated
- To support record level operations, we must:
  - keep track of the *pages* in a file
  - keep track of *free space* on pages
  - keep track of the *records* on a page
- There are many alternatives for keeping track of this.
  - We'll consider 2

Heap File Implemented as a List



- The header page id and Heap file name must be stored someplace.
  - Database "catalog"
- Each page contains 2 `pointers' plus data.

Heap File Using a Page Directory

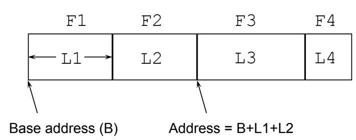


- The entry for a page can include the number of free bytes on the page.
- The directory is a collection of pages; linked list implementation is just one alternative.
  - *Much smaller than linked list of all HF pages*!

Indexes (a sneak preview)

- 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 a gpa > 3
- Indexes are file structures that enable us to answer such value-based queries efficiently.

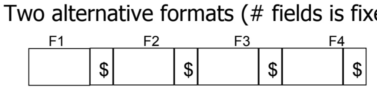
Record Formats: Fixed Length



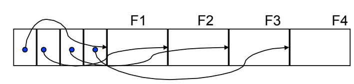
- Information about field types same for all records in a file; stored in *system catalogs.*
- Finding *i'th* field done via arithmetic.

Record Formats: Variable Length

- Two alternative formats (# fields is fixed):



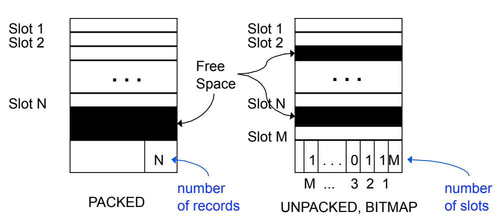
Fields Delimited by Special Symbols



Array of Field Offsets

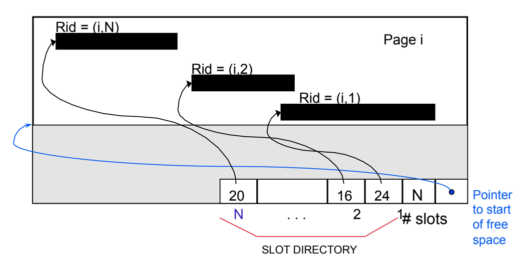
☛ Second offers direct access to i'th field, efficient storage of nulls (special *don't know* value); small directory overhead.

Page Formats: Fixed Length Records



☛ *Record id = <page id, slot #>. In first alternative, moving records for free space management changes rid; may not be acceptable.*

Page Formats: Variable Length Records



☛ *Can move records on page without changing rid; so, attractive for fixed-length records too.*

System Catalogs

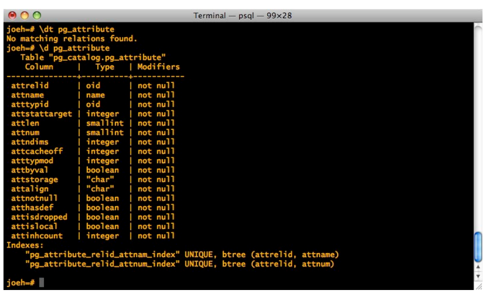
- For each relation:
  - name, file location, file structure (e.g., Heap file)
  - attribute name and type, for each attribute
  - index name, for each index
  - integrity constraints
- For each index:
  - structure (e.g., B+ tree) and search key fields
- For each view:
  - view name and definition
-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

☛ *Catalogs are themselves stored as relations!*

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

pg_attribute



Summary

- Disks provide cheap, non-volatile storage.
  - Better random access than tape, worse than RAM
  - Arrange data to minimize *seek* and *rotation* delays.
    - Depends on workload!
- Buffer manager brings pages into RAM.
  - Page pinned in RAM until released by requestor.
  - Dirty pages written to disk when frame replaced (sometime after requestor unpins the page).
  - Choice of frame to replace based on *replacement policy.*
  - Tries to *pre-fetch* several pages at a time.

Summary (Contd.)

- DBMS vs. OS File Support
  - DBMS needs non-default features
  - Careful timing of writes, control over prefetch
- Variable length record format
  - Direct access to i'th field and null values.
- Slotted page format
  - Variable length records and intra-page reorg

Summary (Contd.)

- DBMS "File" tracks collection of pages, records within each.
  - Pages with free space identified using linked list or directory structure
- Indexes support efficient retrieval of records based on the values in some fields.
- Catalog relations store information about relations, indexes and views.