

## 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*



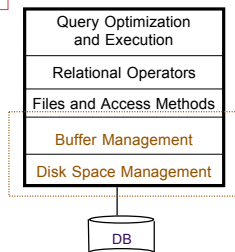
## Review

- Aren't Databases Great?
- Relational model
- SQL



## Disks, Memory, and Files

The BIG picture...



## Disks and Files

- **DBMS stores information on disks.**
  - In an electronic world, disks are a mechanical anachronism!
- **This has major implications for DBMS design!**
  - **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.** For \$1000, Dell will sell you either 2 GB of RAM or 180 GB of disk today.
- **Main memory is volatile.** We want data to be saved between runs. (Obviously!)
- **Typical storage hierarchy:**
  - Main memory (RAM) for currently used data.
  - Disk for the main database (secondary storage).
  - Tapes for archiving older versions of the data (tertiary storage)



## Disks

- **Secondary storage device of choice.**
- **Main advantage over tapes:** *random access* vs. *sequential*.
- **Data is stored and retrieved in units called *disk blocks* or *pages*.**
- **Unlike RAM, time to retrieve a disk block varies depending upon location on disk.**
  - Therefore, 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 between about 0.3 and 10msec
  - Rotational delay varies from 0 to 6msec
  - Transfer rate around .008msec 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 software manages space on disk (using OS file system or not?).**
- **Higher levels call upon this layer to:**
  - allocate/de-allocate a page
  - read/write a page
- **Best if a request for a *sequence* of pages is 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 assume sequential access for files!
    - Hence disk space manager should do a decent job.

### Context

```

  graph TD
    A[Query Optimization and Execution] --> B[Relational Operators]
    B --> C[Files and Access Methods]
    C --> D[Buffer Management]
    D --> E[Disk Space Management]
    E --- F[(DB)]
  
```

### Buffer Management in a DBMS

Page Requests from Higher Levels

- **Data must be in RAM for DBMS to operate on it!**
- **Buffer Mgr 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>*
- **If requested page is not in pool:**
  - Choose a frame for *replacement*.
  - Only "un-pinned" pages are candidates!
  - If frame is "dirty", write it to disk
  - Read requested page into chosen frame
- **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 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.
  - To pin a page, *pin\_count++*
  - A page is a candidate for replacement iff *pin count == 0* ("unpinned")
- **CC & recovery may entail additional I/O when a frame is chosen for 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, etc.
- **Policy can have big impact on # of I/O's; depends on the access pattern.**

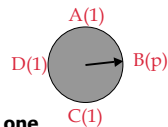


## LRU Replacement Policy

- **Least Recently Used (LRU)**
  - for each page in buffer pool, keep track of time when last *unpinned*
  - replace the frame which has the oldest (earliest) time
  - very common policy: intuitive and simple
    - Works well for repeated accesses to popular pages
- **Problems?**
- **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;
  else if (pincount == 0 && ref bit is off)
    choose this page for replacement;
} until a page is chosen;

```

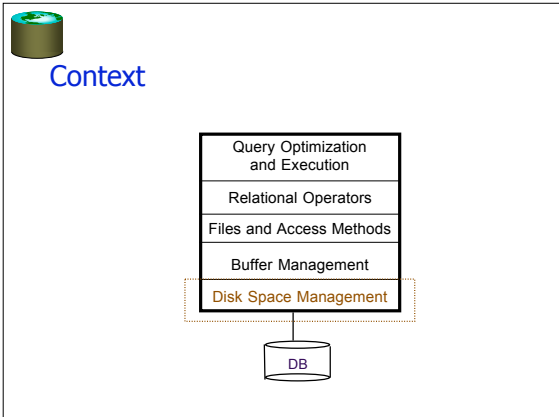
Questions:  
How like LRU?  
Problems?



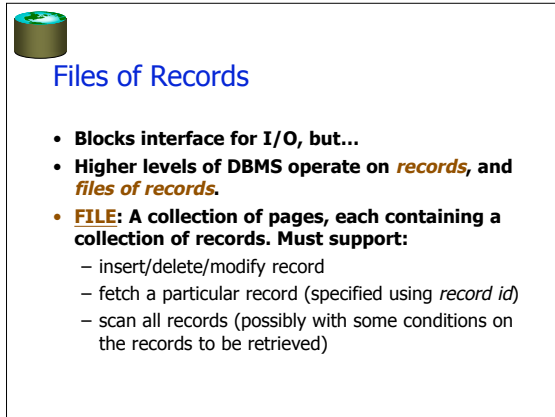
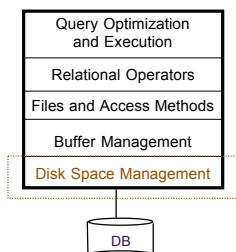
## DBMS vs. OS File System

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

- **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 & *order writes* (important for implementing CC & recovery)
  - adjust *replacement policy*, and *pre-fetch pages* based on access patterns in typical DB operations.

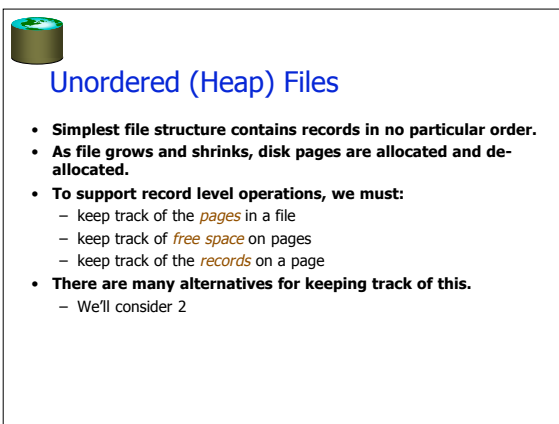


## Context



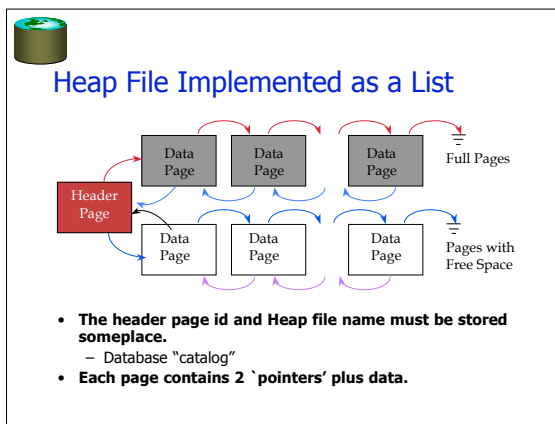
## Files of Records

- **Blocks 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)



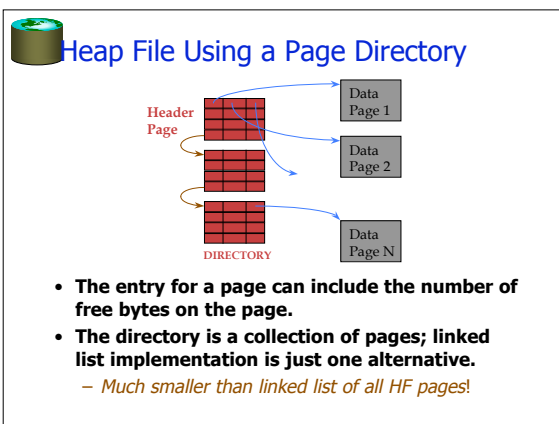
## Unordered (Heap) Files

- **Simplest file structure contains records in no particular order.**
- **As 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.**
  - We'll consider 2

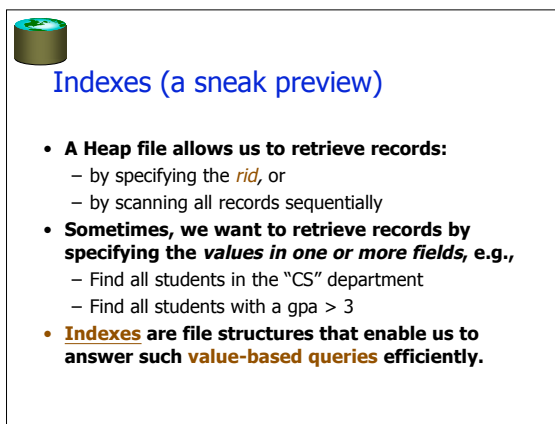


## Heap File Implemented as a List

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



- **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):

- 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

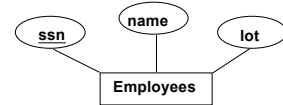


## The ER Model, a brief tangent

- Many Data Models, ER is one of them
- Easier for people to use than Relational
  - More expressive
  - Graphical representation
- Converts fairly easily to Relational Schema



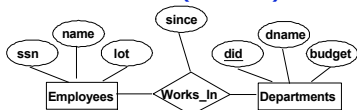
## ER Model Basics



- **Entity:** Real-world object distinguishable from other objects. An entity is described (in DB) using a set of **attributes**.
- **Entity Set:** A collection of similar entities. E.g., all employees.
  - All entities in an entity set have the same set of attributes. (Until we consider hierarchies, anyway!)
  - Each entity set has a **key**.
  - Each attribute has a **domain**.



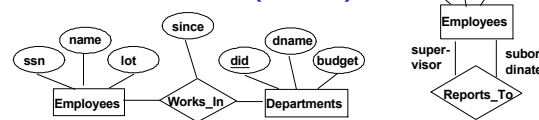
## ER Model Basics (Contd.)



- **Relationship:** Association among two or more entities. E.g., Jones works in Pharmacy department.
- **Relationship Set:** Collection of similar relationships.
  - An n-ary relationship set R relates n entity sets E1 ... En; each relationship in R involves entities e1 E1, ..., en En



## ER Model Basics (Contd.)

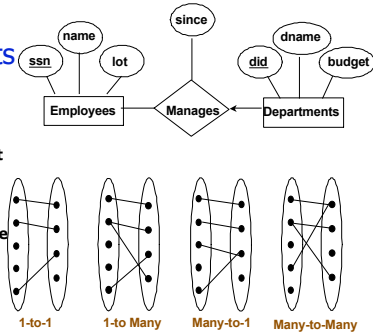


- Same entity set could participate in different relationship sets, or in different "roles" in same set.



## Key Constraints

- Consider Works\_In: An employee can work in many departments; a dept can have many employees.
- In contrast, each dept has at most one manager, according to the **key constraint** on Manages.



## ER Summary

- Define schema with ER Diagram
- Key-foreign key relationships more explicit
- Structure of DBMS easier for people to understand



## 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*.
  - Tries to *pre-fetch* several pages at a time.



## Summary (Contd.)

- **DBMS vs. OS File Support**
  - DBMS needs features not found in many OS's, 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.*)**