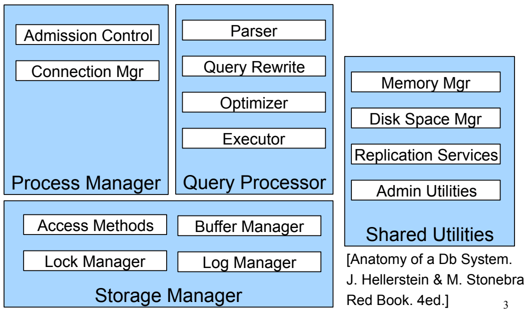
CSE 444: Database Internals

Lecture 4
Data storage and buffer management

Important Note

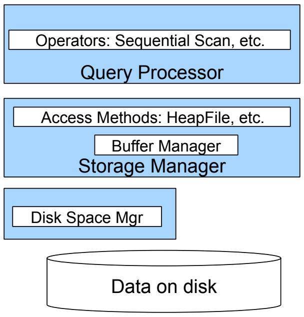
- Lectures show principles
- You need to think through what you will actually implement in SimpleDB!
  - Try to implement the simplest solutions
- If you are confused between the lectures and the labs, tell us!

DBMS Architecture



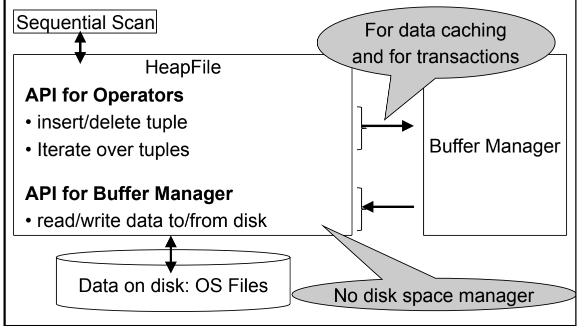
[Anatomy of a Db System.
J. Hellerstein & M. Stonebraker.
Red Book. 4ed.]

Today: Starting at the Bottom



- **Operators:** Process data
- **Access methods**: Organize data to support fast access to desired subsets of records
- **Buffer manager**: Caches data in memory. Reads/ writes data to/from disk as needed
- **Disk-space manager**: Allocates space on disk for files/access methods

HeapFile In SimpleDB

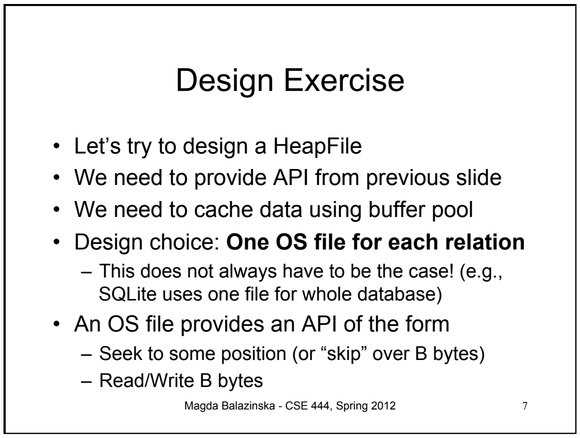


API for Operators
- insert/delete tuple
- Iterate over tuples

API for Buffer Manager
- read/write data to/from disk

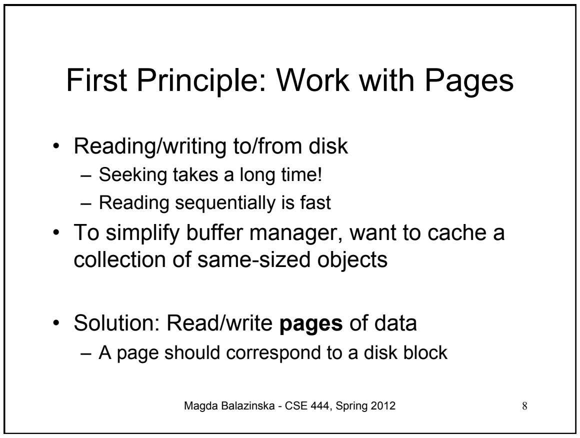
General HeapFile Operations

- **Create** or **destroy** a file
- **Insert** a record
- **Delete** a record with a given rid (rid)
  - rid: unique tuple identifier (more later)
- **Get** a record with a given rid
  - Not necessary for sequential scan operator
  - But used with indexes (more next lecture)
- **Scan** all records in the file



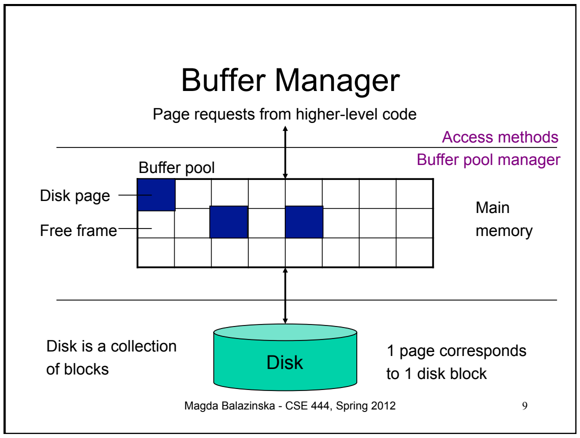
Design Exercise

- Let's try to design a HeapFile
- We need to provide API from previous slide
- We need to cache data using buffer pool
- Design choice: **One OS file for each relation**
  - This does not always have to be the case! (e.g., SQLite uses one file for whole database)
- An OS file provides an API of the form
  - Seek to some position (or "skip" over B bytes)
  - Read/Write B bytes



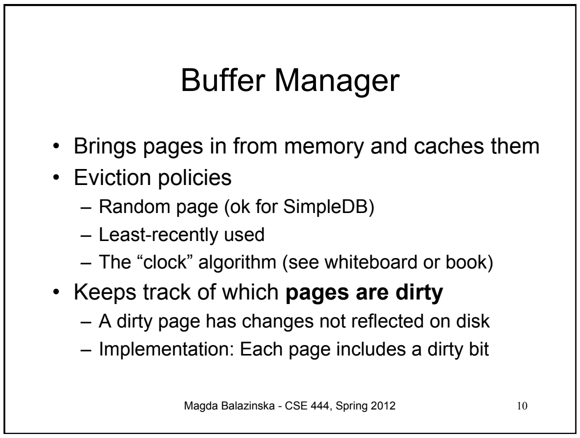
First Principle: Work with Pages

- Reading/writing to/from disk
  - Seeking takes a long time!
  - Reading sequentially is fast
- To simplify buffer manager, want to cache a collection of same-sized objects
- Solution: Read/write **pages** of data
  - A page should correspond to a disk block



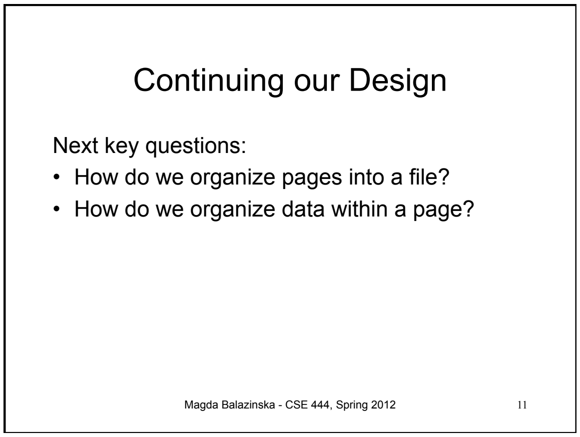
Buffer Manager

Page requests from higher-level code
Access methods
Buffer pool manager
Buffer pool
Disk page
Free frame
Main memory
Disk
Disk is a collection of blocks
1 page corresponds to 1 disk block



Buffer Manager

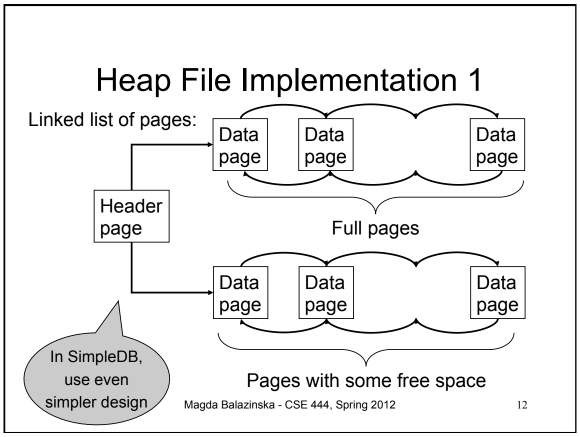
- Brings pages in from memory and caches them
- Eviction policies
  - Random page (ok for SimpleDB)
  - Least-recently used
  - The "clock" algorithm (see whiteboard or book)
- Keeps track of which **pages are dirty**
  - A dirty page has changes not reflected on disk
  - Implementation: Each page includes a dirty bit



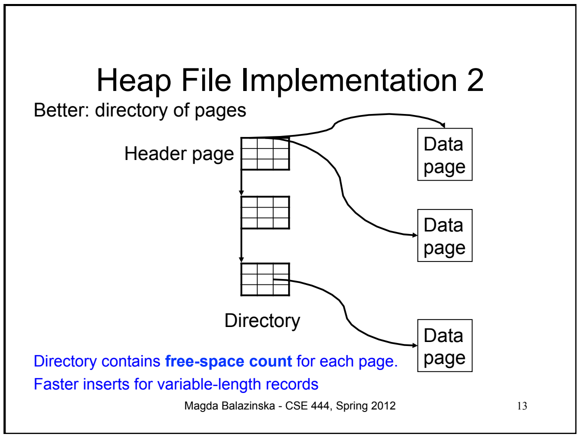
Continuing our Design

Next key questions:

- How do we organize pages into a file?
- How do we organize data within a page?



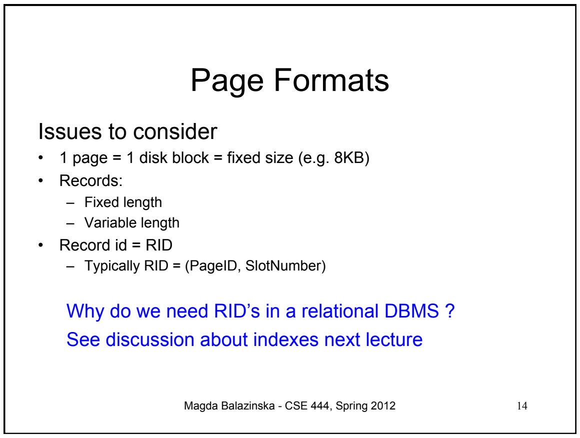
Heap File Implementation 1



Heap File Implementation 2

Better: directory of pages

Directory contains **free-space count** for each page.
Faster inserts for variable-length records

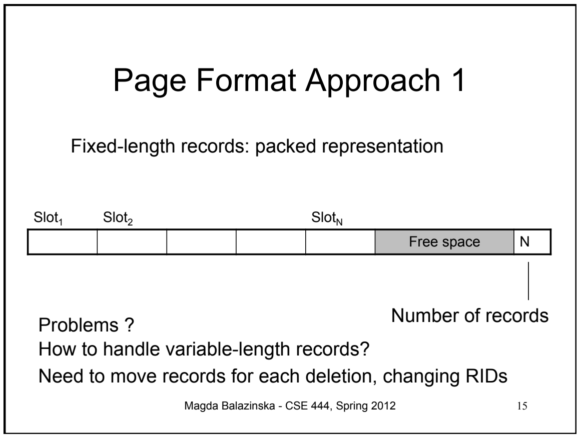


Page Formats

Issues to consider

- 1 page = 1 disk block = fixed size (e.g. 8KB)
- Records:
  - Fixed length
  - Variable length
- Record id = RID
  - Typically RID = (PageID, SlotNumber)

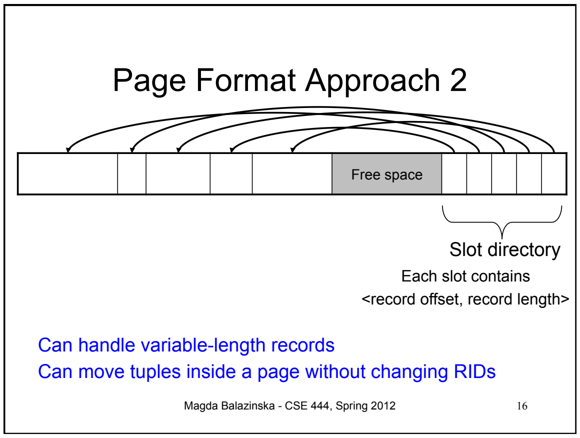
Why do we need RID's in a relational DBMS ?
See discussion about indexes next lecture



Page Format Approach 1

Fixed-length records: packed representation

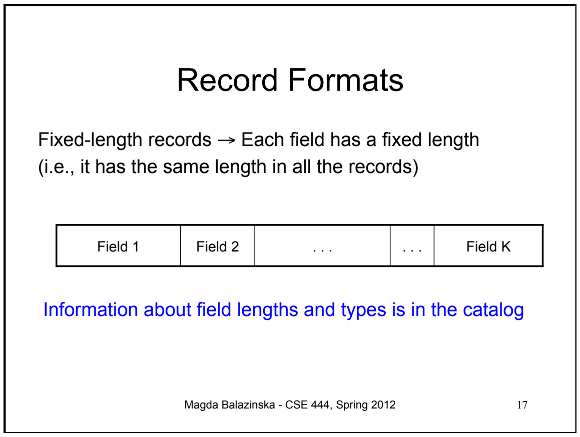
Problems ?
How to handle variable-length records?
Need to move records for each deletion, changing RIDs



Page Format Approach 2

Each slot contains
<record offset, record length>

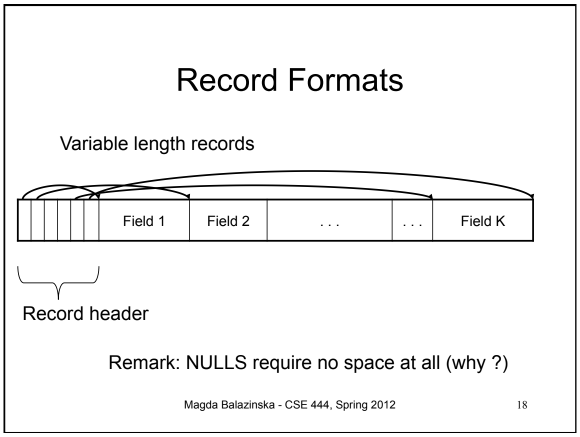
Can handle variable-length records
Can move tuples inside a page without changing RIDs



Record Formats

Fixed-length records → Each field has a fixed length
(i.e., it has the same length in all the records)

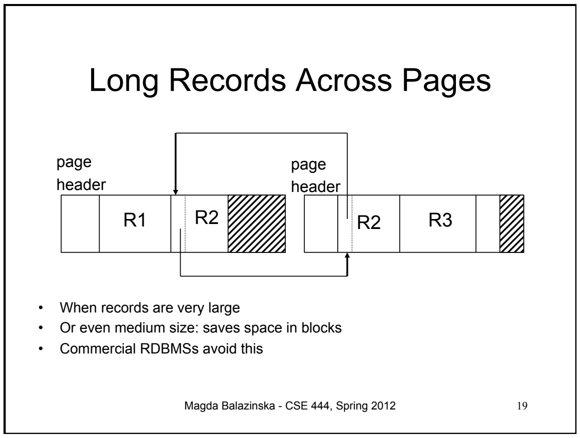
Information about field lengths and types is in the catalog



Record Formats

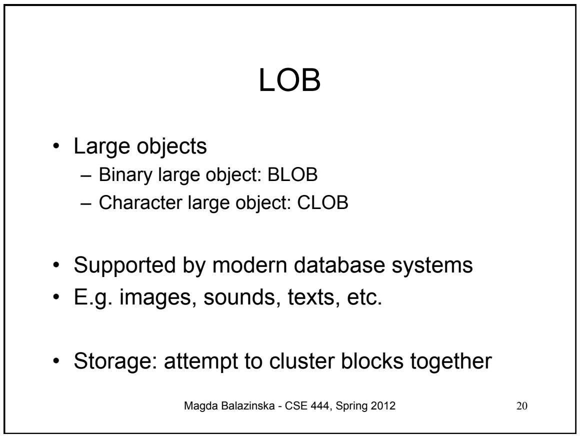
Variable length records

Remark: NULLS require no space at all (why ?)



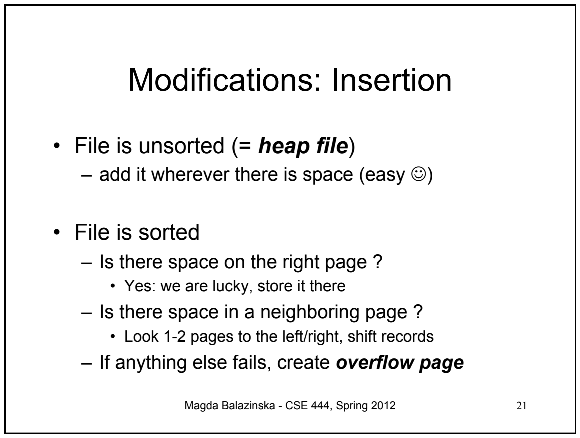
Long Records Across Pages

- When records are very large
- Or even medium size: saves space in blocks
- Commercial RDBMSs avoid this



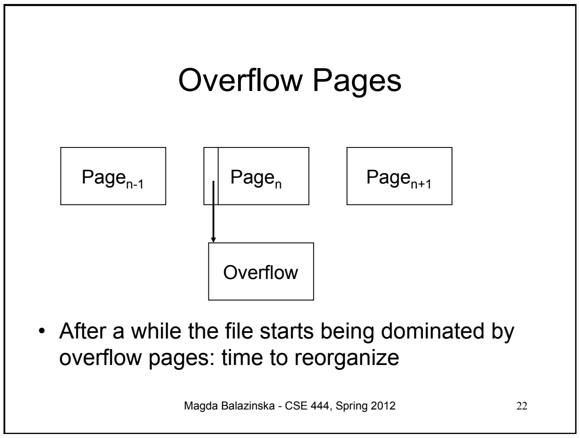
LOB

- Large objects
  - Binary large object: BLOB
  - Character large object: CLOB
- Supported by modern database systems
- E.g. images, sounds, texts, etc.
- Storage: attempt to cluster blocks together



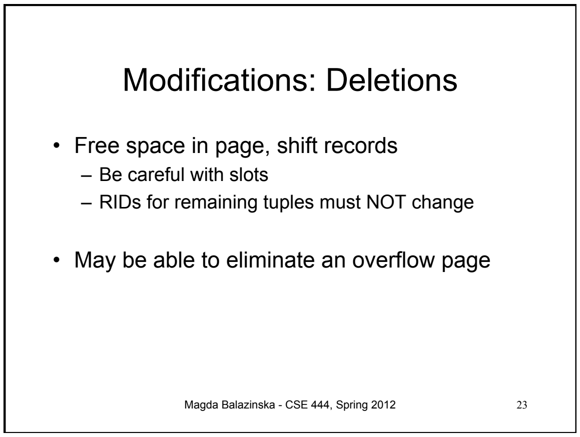
Modifications: Insertion

- File is unsorted (= ***heap file***)
  - add it wherever there is space (easy ☺)
- File is sorted
  - Is there space on the right page ?
    - Yes: we are lucky, store it there
  - Is there space in a neighboring page ?
    - Look 1-2 pages to the left/right, shift records
  - If anything else fails, create ***overflow page***



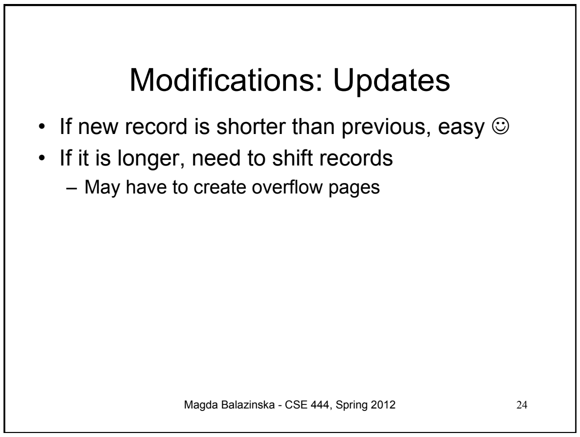
Overflow Pages

- After a while the file starts being dominated by overflow pages: time to reorganize



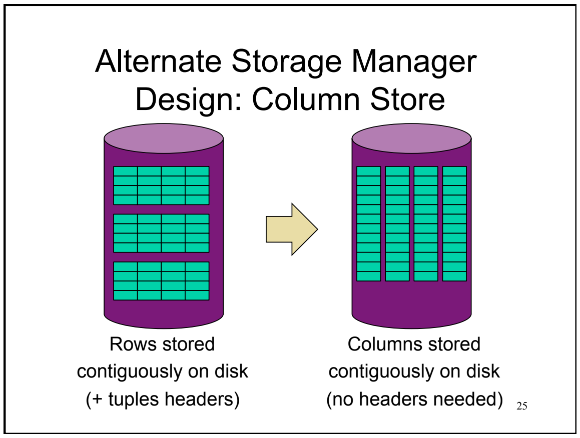
Modifications: Deletions

- Free space in page, shift records
  - Be careful with slots
  - RIDs for remaining tuples must NOT change
- May be able to eliminate an overflow page



Modifications: Updates

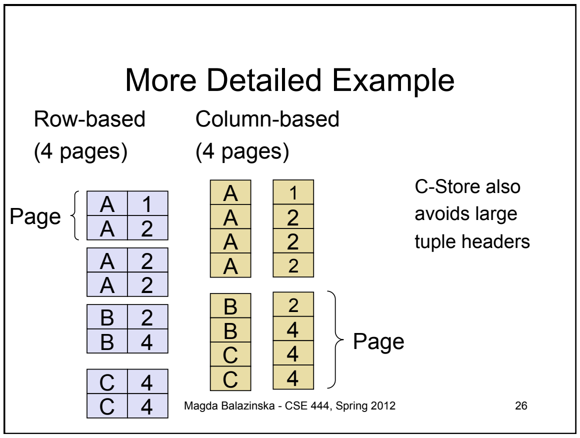
- If new record is shorter than previous, easy ☺
- If it is longer, need to shift records
  - May have to create overflow pages



Alternate Storage Manager Design: Column Store

Rows stored contiguously on disk (+ tuples headers)

Columns stored contiguously on disk (no headers needed)



More Detailed Example

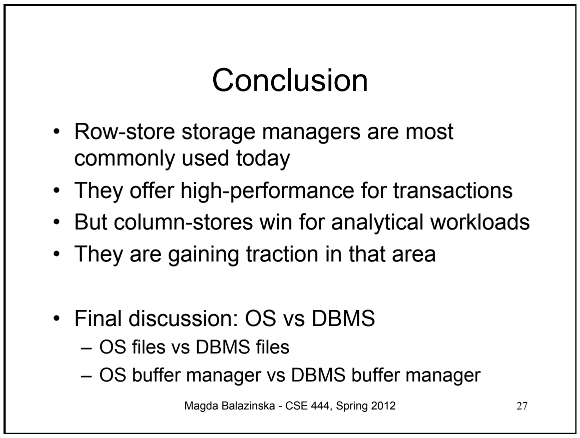
Row-based (4 pages)

A	1
A	2
A	2
A	2
B	2
B	4
C	4
C	4

Column-based (4 pages)

A	1
A	2
A	2
A	2
B	2
B	4
C	4
C	4

C-Store also avoids large tuple headers



Conclusion

- Row-store storage managers are most commonly used today
- They offer high-performance for transactions
- But column-stores win for analytical workloads
- They are gaining traction in that area

- Final discussion: OS vs DBMS
  - OS files vs DBMS files
  - OS buffer manager vs DBMS buffer manager