

# CSE 444: Database Internals

## Lecture 4 Data storage and (more) buffer management

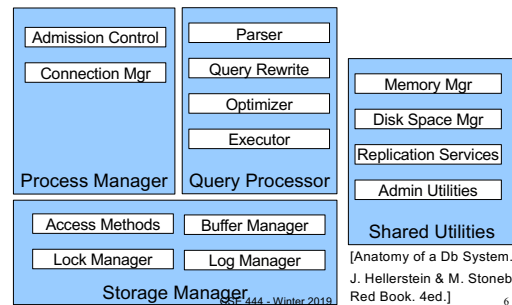
# Announcements

- Lab 1 Part 1 due today:
  - Need to run the `.turnInLab.sh` script with the tag
- Homework 1 due Friday
  - Very helpful for rest of lab 1, don't wait until Thursday to start!
- Anyone still without a repo should email me

# Important Note

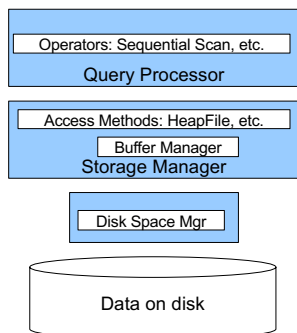
- Lectures show principles
- You need to think through what you will actually implement in SimpleDB!
  - Try to implement the simplest solutions
- If you are confused, tell us!
  - Thursday section this week will be extra lab help

# DBMS Architecture



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

# DBMS Architecture



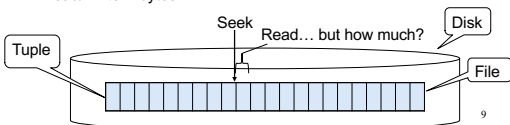
# Today: Starting at the Bottom

Consider a relation storing tweets:  
`Tweets(tid, user, time, content)`

How should we store it on disk?

## Design Exercise

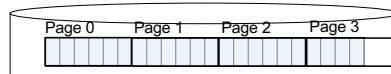
- One design choice: **One OS file for each relation**
  - This does not always have to be the case! (e.g., SQLite uses one file for whole database)
  - DBMSs can also use disk drives directly
- An OS file provides an API of the form
  - Seek to some position (or "skip" over B bytes)
  - Read/Write B bytes



9

## First Principle: Work with Pages

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



10

## Continuing our Design

Key questions:

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

First, **how could we store some tuples on a page?**

Let's first assume all tuples are of the same size:

```
Tweets(tid int, user char(10),
        time int, content char(140))
```

CSE 444 - Winter 2019

11

## 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 future discussion on indexes and transactions

CSE 444 - Winter 2019

12

## Design Exercise

- Think how you would store tuples on a page
  - Fixed length tuples
  - Variable length tuples
- Compare your solution with your neighbor's

CSE 444 - Winter 2019

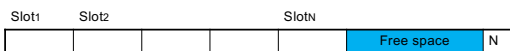
13

## Page Format Approach 1

Fixed-length records: packed representation

Divide page into slots. Each slot can hold one tuple

Record ID (RID) for each tuple is (PageID, SlotNb)



How do we insert a new record?

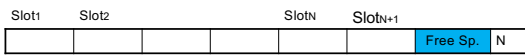
Number of records

CSE 444 - Winter 2019

14

## Page Format Approach 1

Fixed-length records: packed representation  
 Divide page into slots. Each slot can hold one tuple  
 Record ID (RID) for each tuple is (PageID, SlotNb)

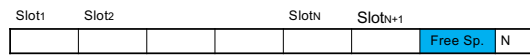


How do we insert a new record?

Number of records

## Page Format Approach 1

Fixed-length records: packed representation  
 Divide page into slots. Each slot can hold one tuple  
 Record ID (RID) for each tuple is (PageID, SlotNb)



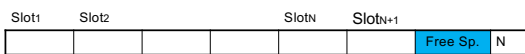
How do we insert a new record?

Number of records

How do we delete a record?

## Page Format Approach 1

Fixed-length records: packed representation  
 Divide page into slots. Each slot can hold one tuple  
 Record ID (RID) for each tuple is (PageID, SlotNb)



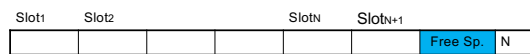
How do we insert a new record?

Number of records

How do we delete a record? What is the problem?

## Page Format Approach 1

Fixed-length records: packed representation  
 Divide page into slots. Each slot can hold one tuple  
 Record ID (RID) for each tuple is (PageID, SlotNb)



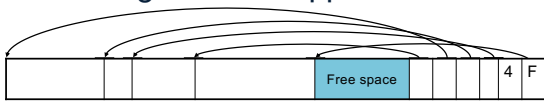
How do we insert a new record?

Number of records

How do we delete a record? Cannot move records! (Why?)

How do we handle variable-length records?

## Page Format Approach 2



Header contains slot directory

+ Need to keep track of # of slots

+ Also need to keep track of free space (F)

Slot directory  
 Each slot contains  
 <record offset, record length>

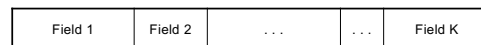
Can handle variable-length records

Can move tuples inside a page without changing RIDs

RID is (PageID, SlotID) combination

## Record Formats

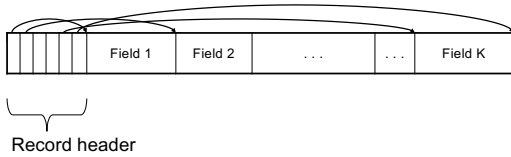
Fixed-length records => Each field has a fixed length  
 (i.e., it has the same length in all the records)



Information about field lengths and types is in the catalog

## Record Formats

Variable length records



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

## 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

## Continuing our Design

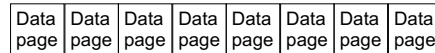
Our key questions:

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

Now, [how should we group pages into files?](#)

## Heap File Implementation 1

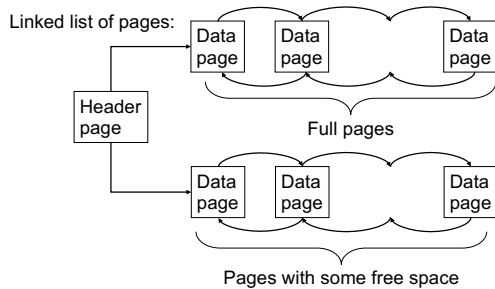
A sequence of pages (implementation in SimpleDB)



Some pages have space and other pages are full  
Add pages at the end when need more space

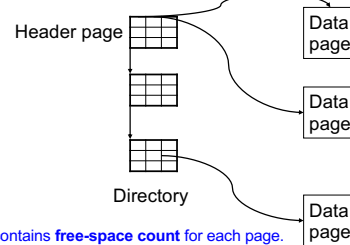
Works well for small files  
But finding free space requires scanning the file...

## Heap File Implementation 2



## Heap File Implementation 3

Better: directory of pages



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

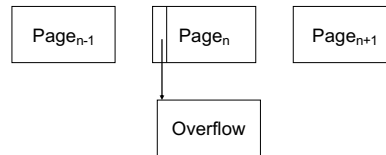
## Modifications: Insertion

- File is unsorted (= **heap file**)
  - add it wherever there is space (easy ☺)
  - add more pages if out of space
- 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**

CSE 444 - Winter 2019

28

## Overflow Pages



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

CSE 444 - Winter 2019

29

## Modifications: Deletions

- Free space by shifting records within page
  - Be careful with slots
  - RIDs for remaining tuples must NOT change
- May be able to eliminate an overflow page

CSE 444 - Winter 2019

30

## Modifications: Updates

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

CSE 444 - Winter 2019

31

## Continuing our Design

We know how to store tuples on disk in a heap file

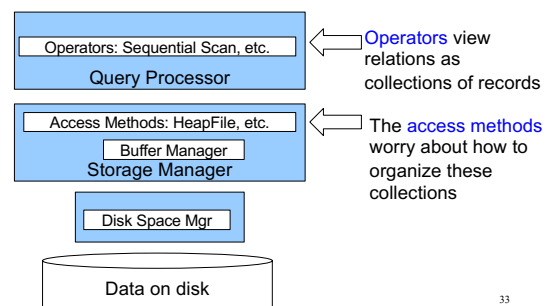
How do these files interact with rest of engine?

- Let's look back at lecture 3

CSE 444 - Winter 2019

32

## How Components Fit Together



33

## Heap File Access Method API

- **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

CSE 444 - Winter 2019 34

## Query Execution How it all Fits Together

(On the fly)  $\pi_{sname}$  **next()**

(On the fly)  $\sigma_{sscity='Seattle' \wedge sstate='WA' \wedge pno=2}$  **next()**

(Nested loop)  $sno = sno$  **next()**

**next()** Suppliers (File scan) **next()** Suppliers (File scan)

35

## Query Execution In SimpleDB

**SeqScan** **open()** **next()** Operator at bottom of plan

In SimpleDB, SeqScan can find HeapFile in Catalog

**Heap File Access Method** **open()** **next()**

- Offers iterator interface
- open()
- next()
- close()
- Knows how to read/write pages from disk

But if Heap File reads data directly from disk, it will not stay cached in Buffer Pool!

36

## Query Execution In SimpleDB

**Everyone shares a single cache**

**HeapFile** Iterator interface: open(), next(), close(). Read/write pages from disk.

**Buffer Pool Manager** handles **getPage()** and **readPage()** requests.

HeapFile2, HeapFile3, HeapFileN

Data on disk: OS Files

Heap files for other relations

37

## Buffer Manager

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

38

## Buffer Manager

Page requests from higher-level code

Access methods, Buffer pool manager

Buffer pool

Disk page, Free frame

Main memory

Disk is a collection of blocks

Disk

1 page corresponds to 1 disk block

39

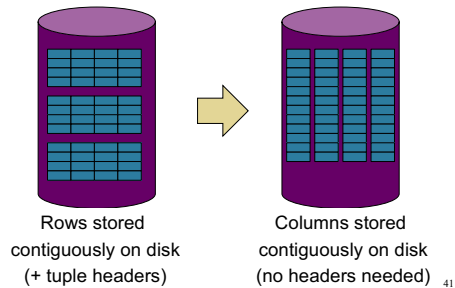
## Pushing Updates to Disk

- When **inserting a tuple**, HeapFile inserts it on a page but does not write the page to disk
- When **deleting a tuple**, HeapFile deletes tuple from a page but does not write the page to disk
- The buffer manager worries when to write pages to disk (and when to read them from disk)
- When need to **add new page** to file, HeapFile adds page to file on disk and then reads it through buffer manager

CSE 444 - Winter 2019

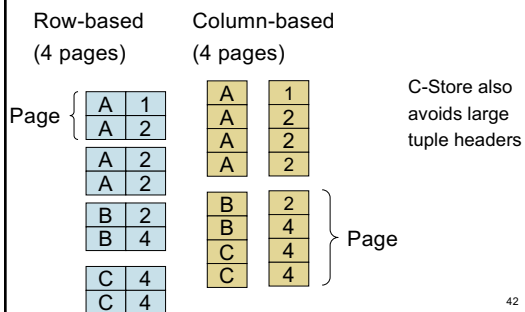
40

## Alternate Storage Manager Design: Column Store



41

## Column Store Illustration



42

## Column Store Example

Sales				
	saleid	product	date	region
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				

Sales				
saleid	product	date	region	
1	1	1	1	
2	2	2	2	
3	3	3	3	
4	4	4	4	
5	5	5	5	
6	6	6	6	
7	7	7	7	
8	8	8	8	
9	9	9	9	
10	10	10	10	

Sales				
saleid	product	date	region	
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				

(a) Column Store with Virtual Ids      (b) Column Store with Explicit Ids      (c) Row Store

Figure 1.1: Physical layout of column-oriented vs row-oriented databases.

The Design and Implementation of Modern Column-Oriented Database Systems Daniel Abadi, Peter Boncz, Stavros Harizopoulos, Stratos Idreos, Samuel Madden, Foundations and Trends® in Databases (Vol 5, Issue 3, 2012, pp 197-260)

CSE 444 - Winter 2019

43

## Conclusion

- Row-store storage managers are most commonly used today for OLTP systems
- They offer high-performance for transactions
- But column-stores win for analytical workloads
- They are widely used in OLAP
- [Optional] Final discussion: OS vs DBMS
  - OS files vs DBMS files
  - OS buffer manager vs DBMS buffer manager

CSE 444 - Winter 2019

44