

CS150A Database

Lu Sun

School of Information Science and Technology

ShanghaiTech University

Mar. 13, 2024

Today:

- Disk Representations:
 - Files, Pages and Records

Readings:

- Database Management Systems (DBMS), Chapter 8
- Lecture note Disk Files

STORING DATA: FILES

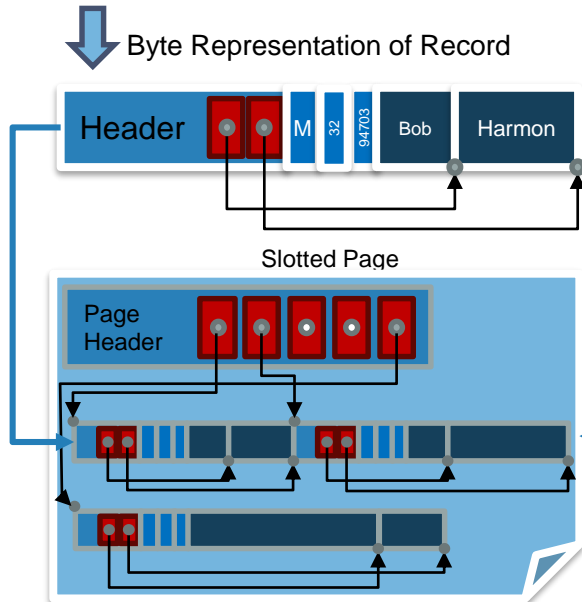
FILE REPRESENTATIONS

Overview: Representations

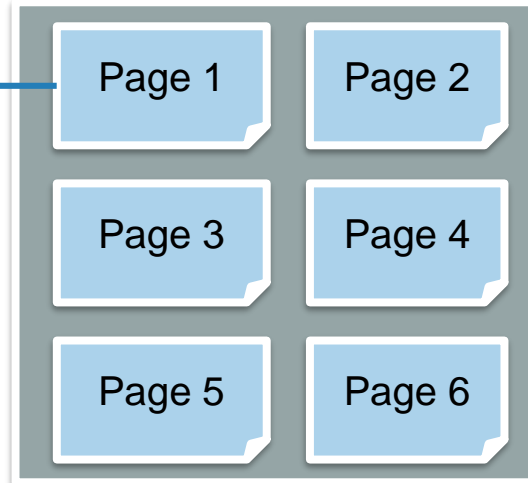
Record

Bob	Harmon	M	32	400
Varchar	Varchar	Char	Int	Int

SSN	Last Name	First Name	Age	Salary
123	Adams	Elmo	31	\$400
443	Grouch	Oscar	32	\$300
244	Oz	Bert	55	\$140
134	Sanders	Ernie	55	\$400

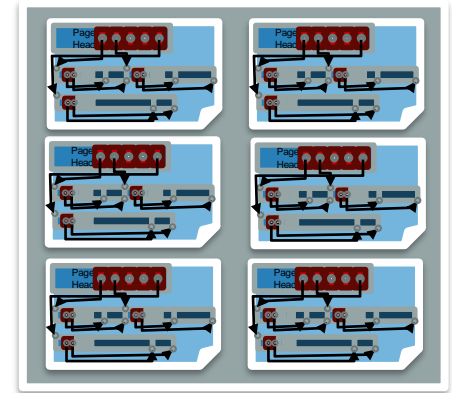


File



Overview: Files of Pages of Records

- Tables stored as logical files
 - Consist of pages
 - Pages contain a collection of records
- Pages are managed
 - On disk by the disk space manager:
pages read/written to physical disk/files
 - In memory by the buffer manager:
higher levels of DBMS only operate in memory



DATABASE FILES

Files of Pages of Records

- **DB FILE**: A collection of pages, each containing a collection of records.
- API for higher layers of the DBMS:
 - Insert/delete/modify record
 - Fetch a particular record by ***record id*** ...
 - Record id is a pointer encoding pair of (**pageID**, **location** on page)
 - Scan all records
 - Possibly with some conditions on the records to be retrieved
- Could span multiple OS files and even machines
 - Or “raw” disk devices

Many DB File Structures

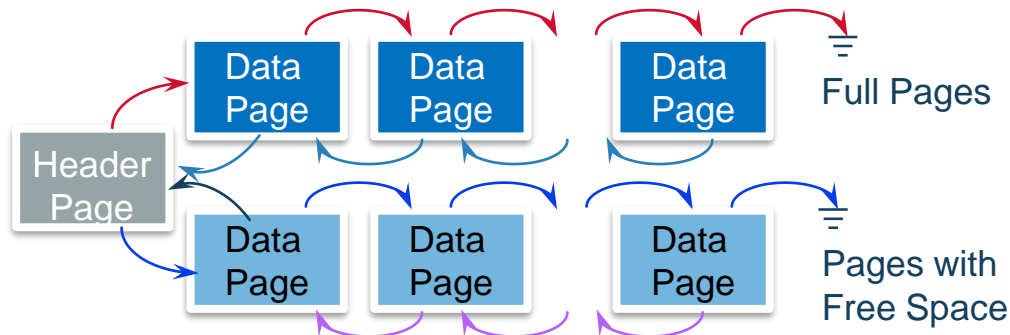
- Unordered Heap Files
 - Records placed arbitrarily across pages
- Clustered Heap Files
 - Records and pages are grouped
- Sorted Files
 - Pages and records are in sorted order
- Index Files
 - B+ Trees, Linear Hashing, ...
 - May contain records or point to records in other files

Unordered Heap Files

- Collection of records in no particular order
 - Not to be confused with “heap” data-structure
- As file shrinks/grows, 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

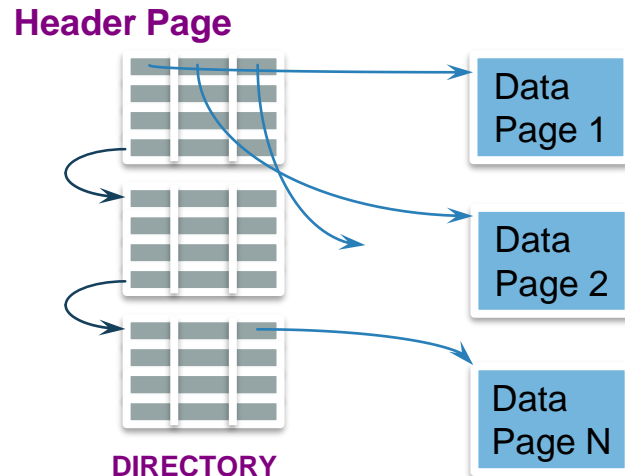
Heap File Implemented as List

- Header page ID and Heap file name stored elsewhere
 - Database catalog
- Each page contains 2 “pointers” plus **free space** and **data**
- What is wrong with this?
 - How do I find a page with enough space for a 20 byte records



Better: Use a Page Directory

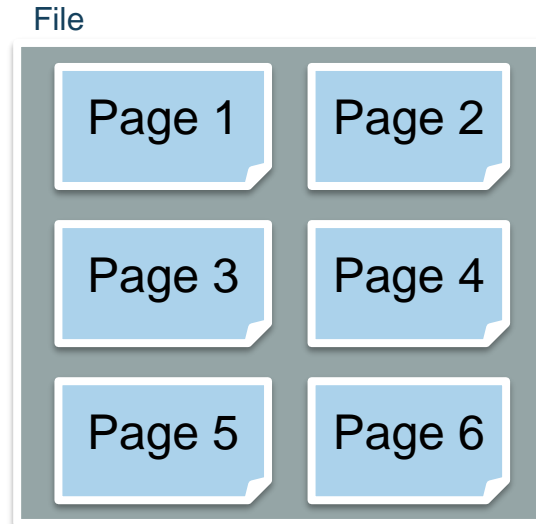
- Directory entries include:
 - #free bytes on the referenced page
- Header pages accessed often → likely in cache
- Finding a page to fit a record required far fewer page loads than linked list
 - Why?
 - One header page load reveals free space of many pages
- You can imagine optimizing the page directory further
 - But diminishing returns?



Summary

- Table encoded as files which are collections of pages

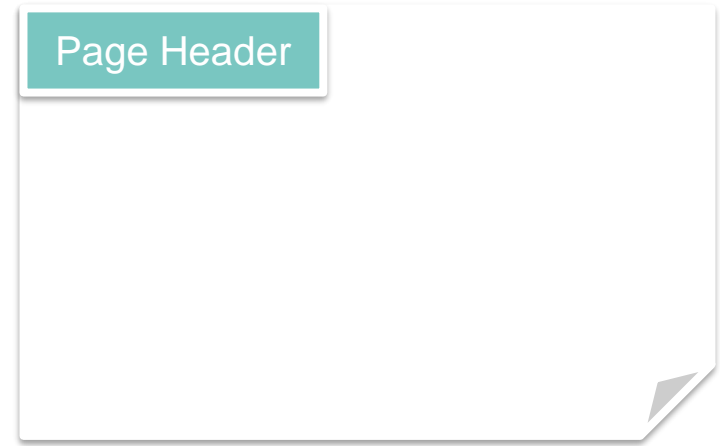
SSNz	Last Name	First Name	Age	Salary
123	Adams	Elmo	31	\$400
443	Grouch	Oscar	32	\$300
244	Oz	Bert	55	\$140
134	Sanders	Ernie	55	\$400



PAGE LAYOUT

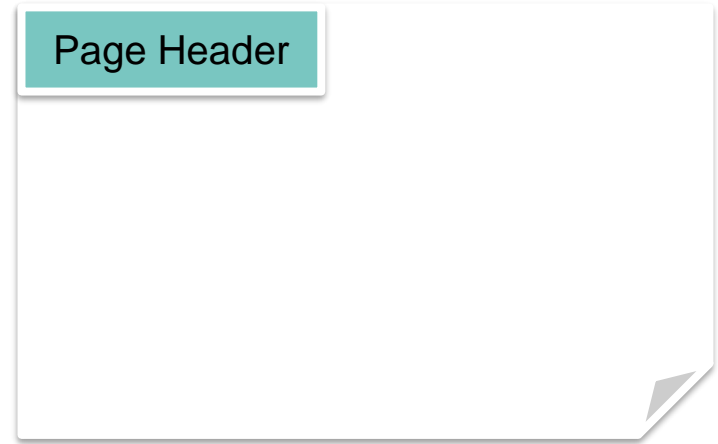
Page Basics: The Header

- Header may contain:
 - Number of records
 - Free space
 - Maybe a next/last pointer
 - Bitmaps, Slot Table



Things to Address

- Record length? Fixed or Variable
- Find records by record id?
 - Record id = (Page id, Location in Page)
- How do we add and delete records?



Options for Page Layouts

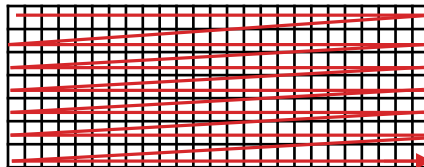
- Depends on
 - Record length (fixed or variable)
 - Page packing (packed or unpacked)

A Note On Imagery

- Data is stored in linear order
 - 1 byte per position
 - Memory addresses are ordered
 - Disk addresses are ordered

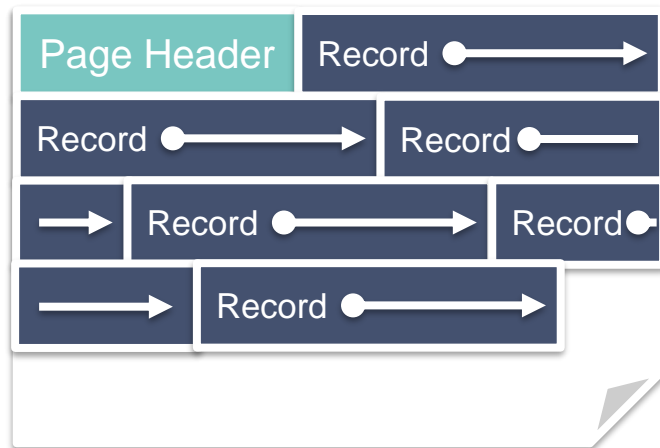


- This doesn't fit nicely on screen
 - So we will “wrap around” the linear order into a rectangle



Fixed Length Records, Packed

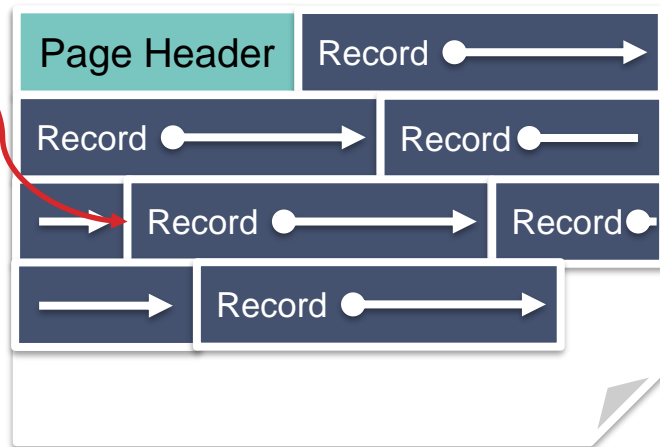
- Pack records densely
- Record id = (pageId, “location in page”)?
 - (pageId, record number in page)!
 - We know the offset from start of page!
- Easy to add: just append
- Delete?



Fixed Length Records, Packed, Pt 2.

- Pack records densely
- Record id = (pageId, “location in page”)?
 - (pageId, record number in page)!
 - We know the offset from start of page!
- Easy to add: just append
- Delete?

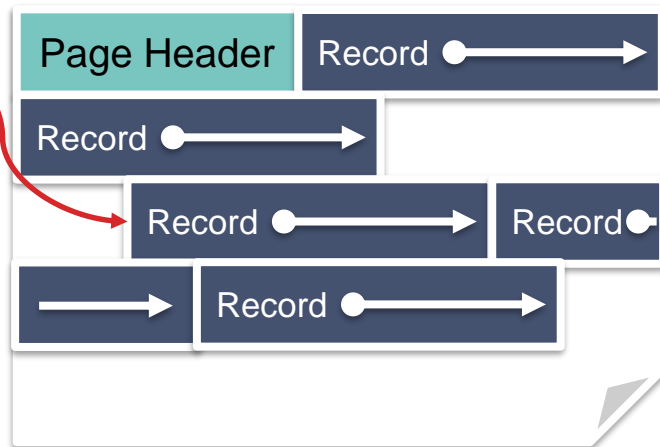
Record id:
(Page 2, Record 4)



Fixed Length Records: Packed, Pt 3.

- Pack records densely
- Record id = (pageId, “location in page”)?
 - (pageId, record number in page)!
 - We know the offset from start of page!
- Easy to add: just append
- Delete?

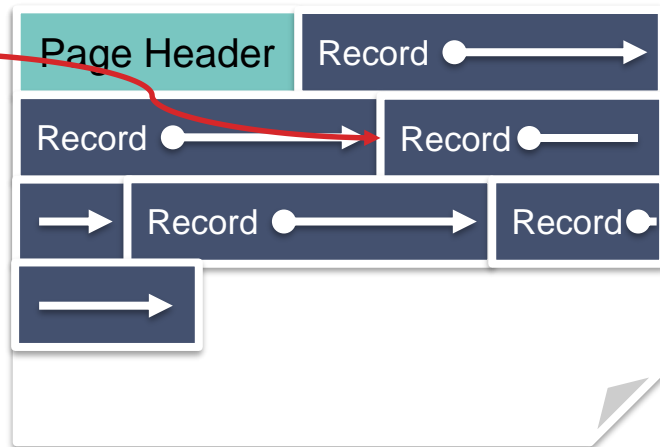
Record id:
(Page 2, Record 4)



Fixed Length Records: Packed, Pt. 4

- Pack records densely
- Record id = (pageId, “location in page”)?
 - (pageId, record number in page)!
 - We know the offset from start of page!
- Easy to add: just append
- Delete?
 - Packed implies re-arrange!
 - Record Id pointers need to be updated!
 - Could be expensive if they’re in other files.

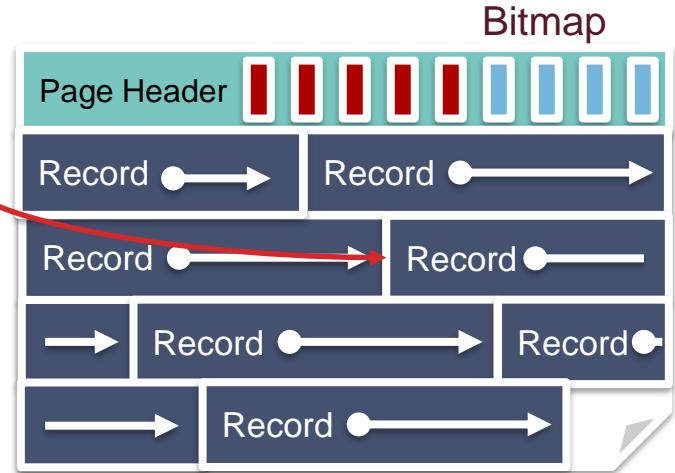
Record id:
(Page 2, Record **3**)



Fixed Length Records: Unpacked

- Bitmap denotes “slots” with records
- Record id: record number in page
- **Insert**: find first empty slot
- **Delete**: Clear bit

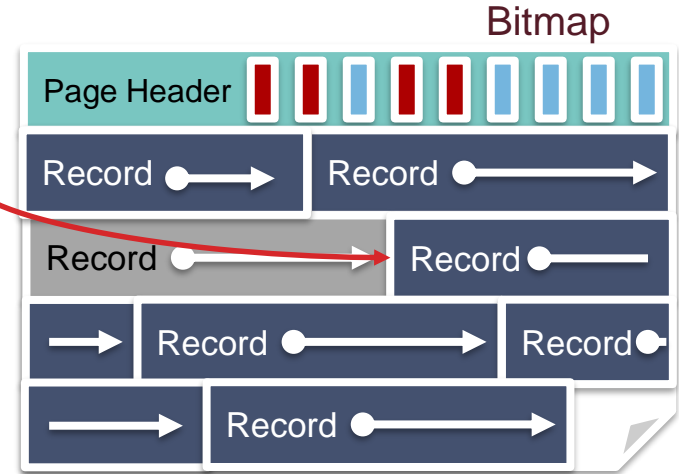
Record id:
(Page 2, Record 4)



Fixed Length Records: Unpacked, Pt. 2

- Bitmap denotes “slots” with records
- Record id: record number in page
- **Insert**: find first empty slot
- **Delete**: Clear bit

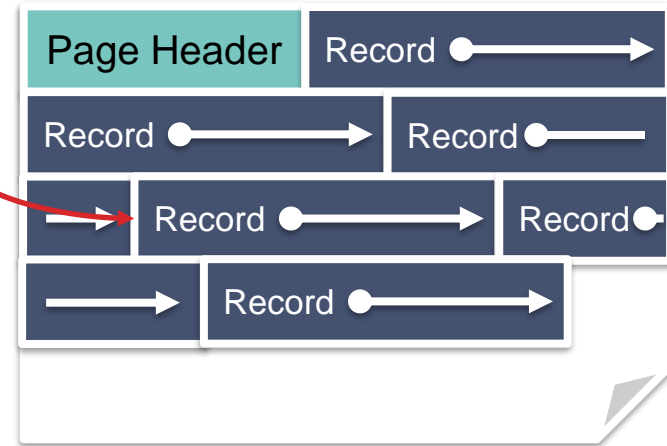
Record id:
(Page 2, Record 4)



Variable Length Records

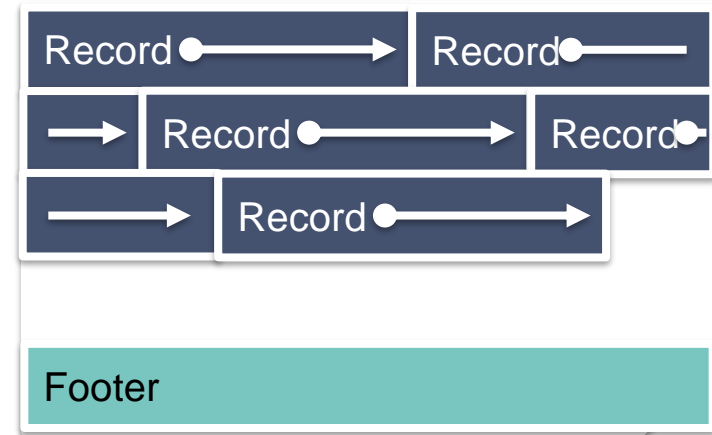
- How do we know where each record begins?
- What happens when we add and delete records?

Record id:
(Page 2, Record 4)



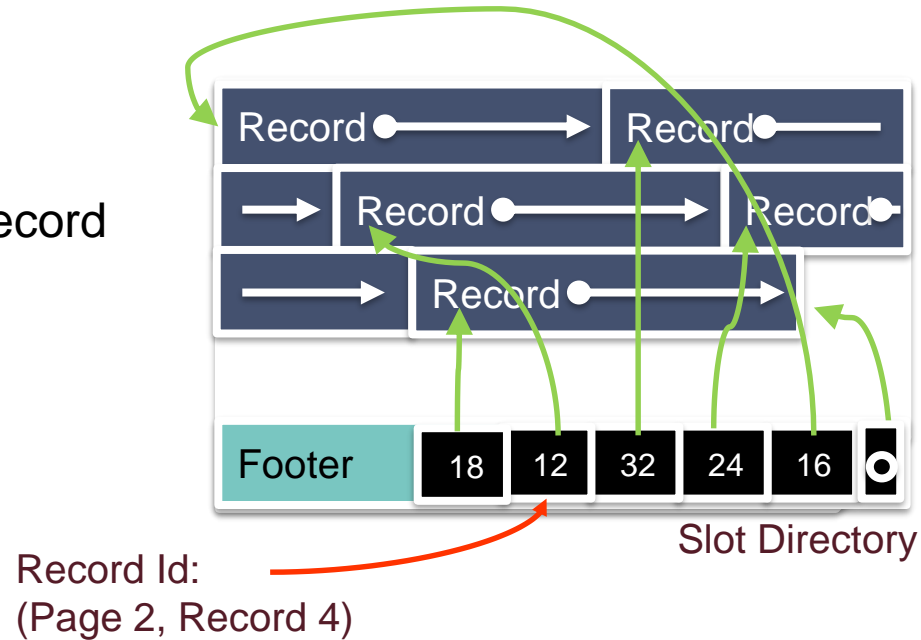
First: Relocate metadata to footer

- We'll see why this is handy shortly...



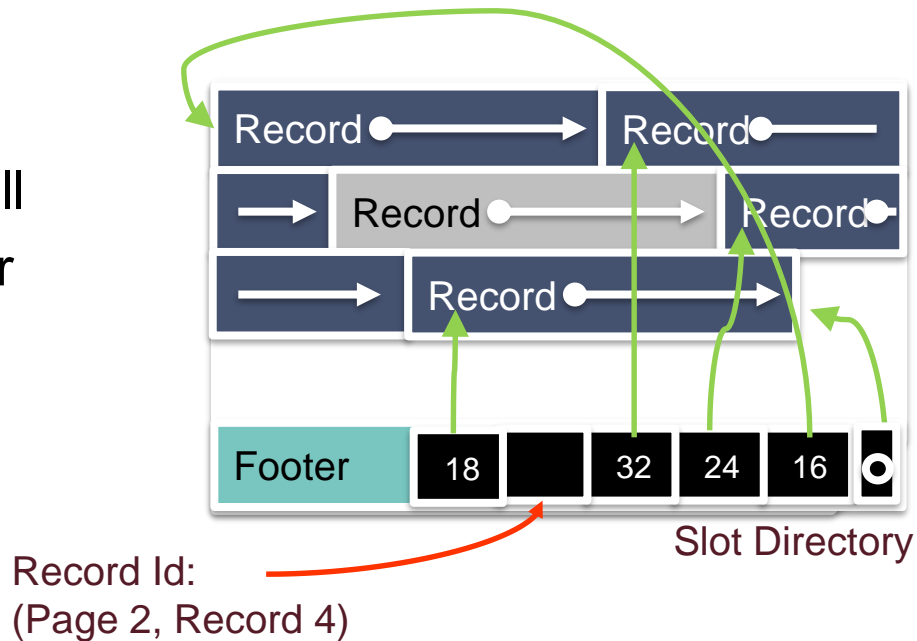
Slotted Page

- Introduce slot directory in footer
 - Pointer to free space
 - Length + Pointer to beginning of record
 - reverse order
- Record ID = location in slot table
 - from right
- Delete?
 - e.g., 4th record on the page



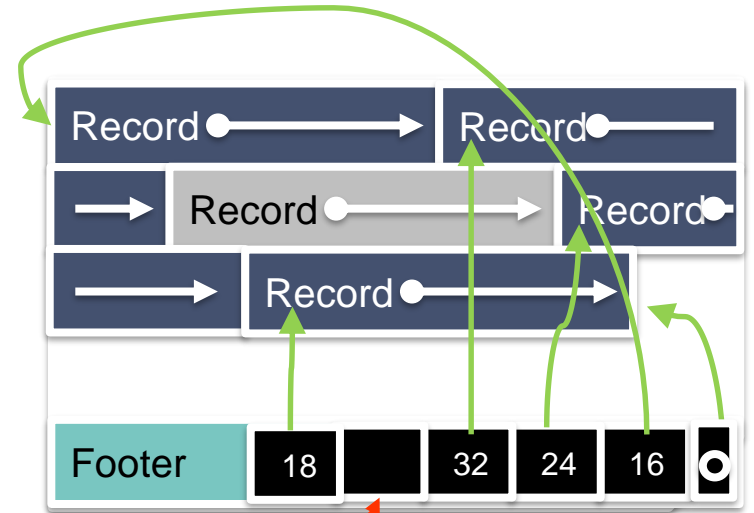
Slotted Page: Delete Record

- Delete record (Page 2, Record 4): Set 4th slot directory pointer to null
 - Doesn't affect pointers to other records



Slotted Page: Insert Record

- Insert:

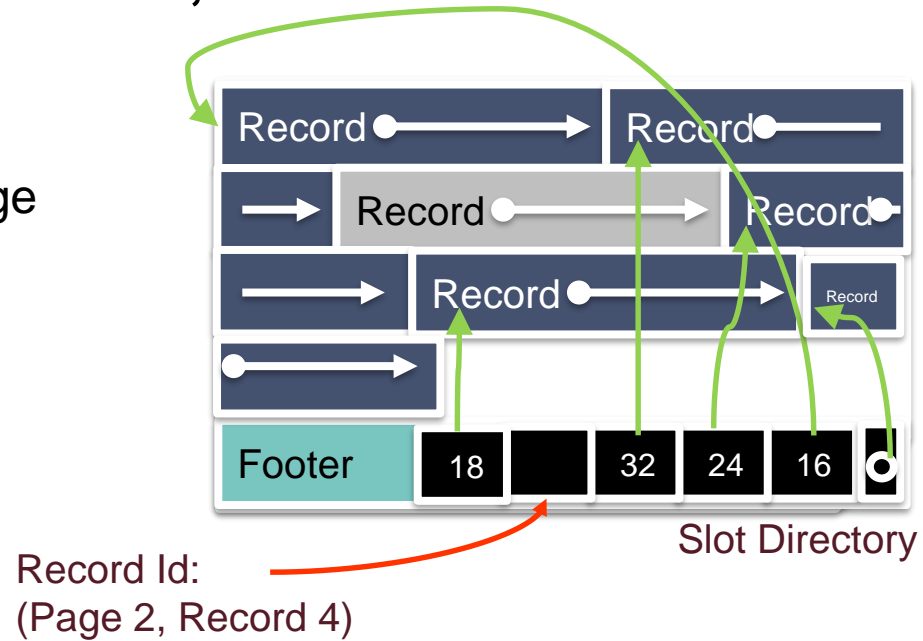


Record Id:
(Page 2, Record 4)

Slot Directory

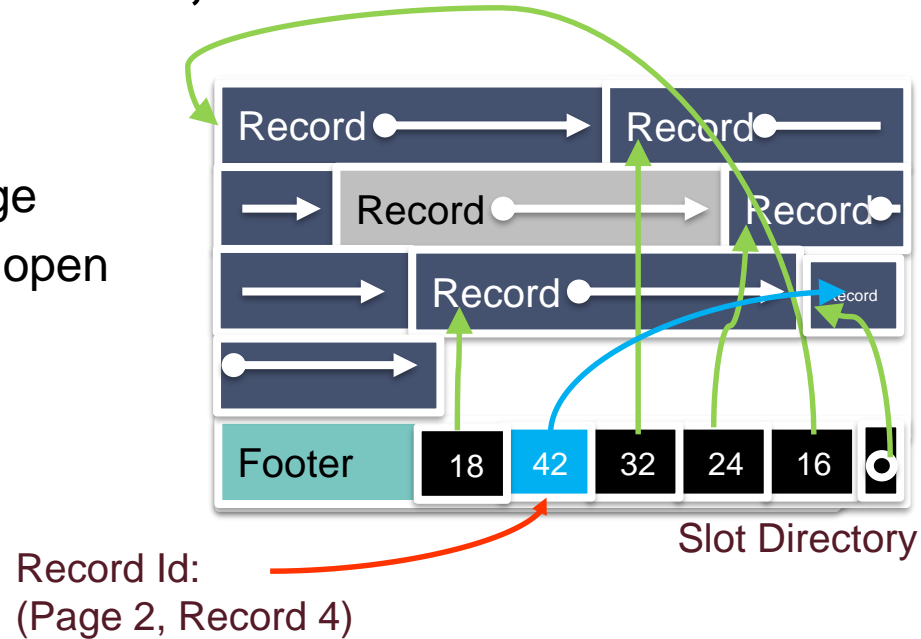
Slotted Page: Insert Record, Pt 2.

- Insert:
 - Place record in free space on page



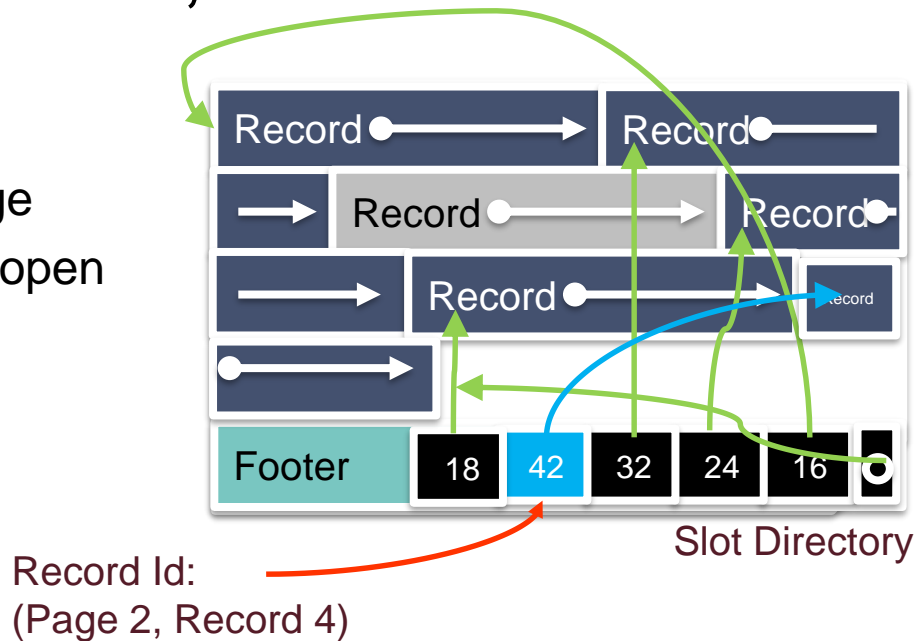
Slotted Page: Insert Record, Pt. 3

- Insert:
 - Place record in free space on page
 - Create pointer/length pair in next open slot in slot directory



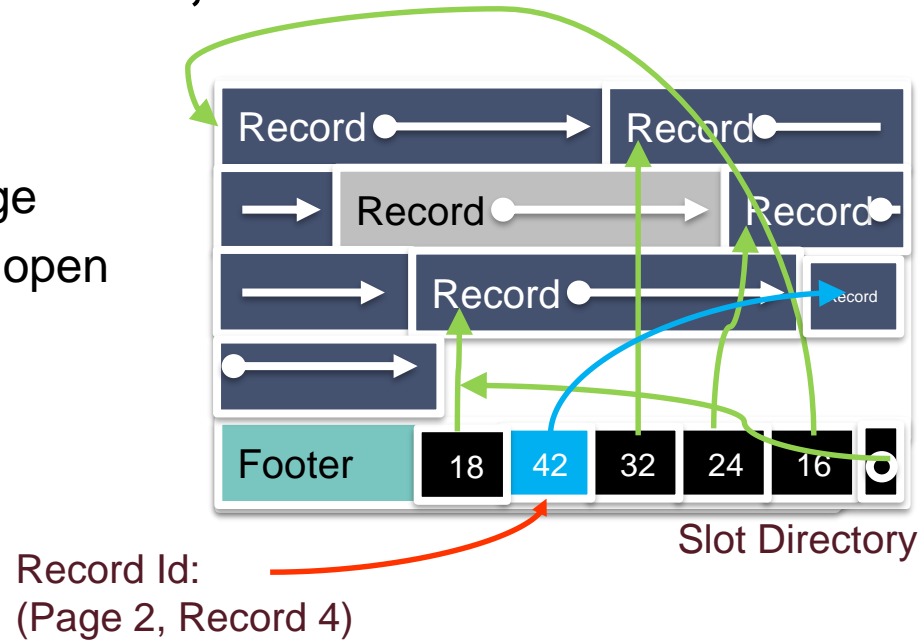
Slotted Page: Insert Record, Pt. 4

- Insert:
 - Place record in free space on page
 - Create pointer/length pair in next open slot in slot directory
 - Update the free space pointer



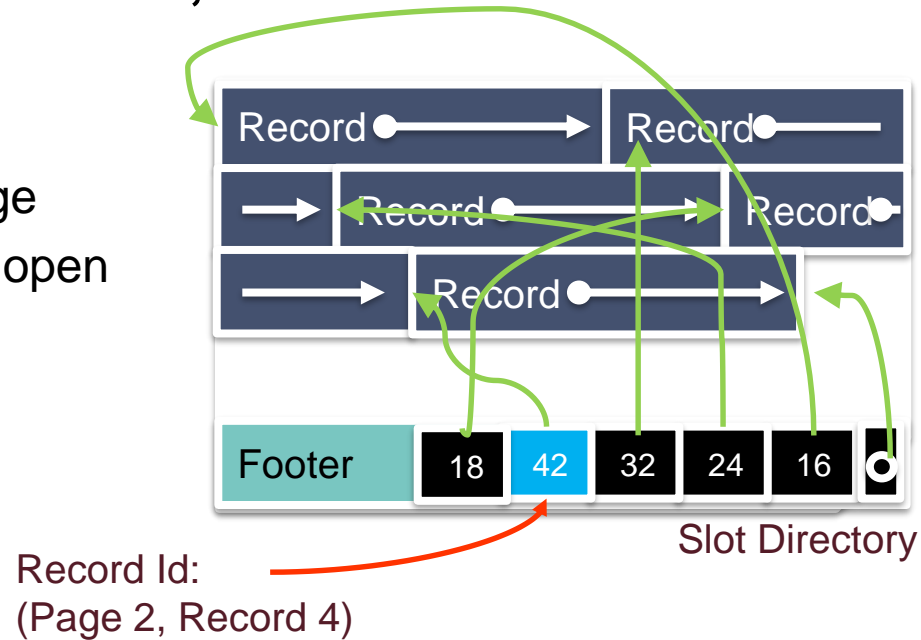
Slotted Page: Insert Record, Pt. 5

- Insert:
 - Place record in free space on page
 - Create pointer/length pair in next open slot in slot directory
 - Update the free space pointer
 - Fragmentation?



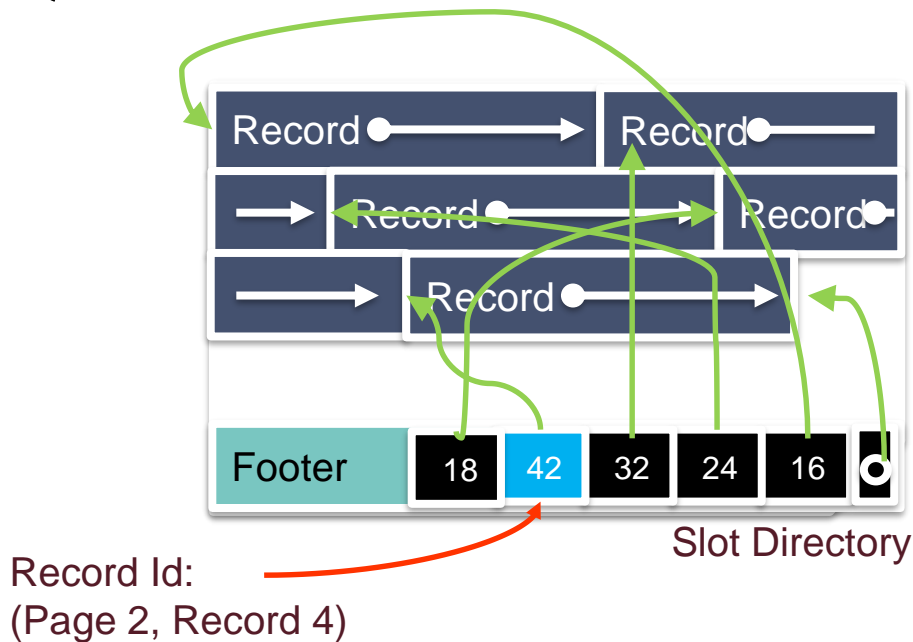
Slotted Page: Insert Record, Pt. 6

- Insert:
 - Place record in free space on page
 - Create pointer/length pair in next open slot in slot directory
 - Update the free space pointer
 - Fragmentation?
 - Reorganize data on page!



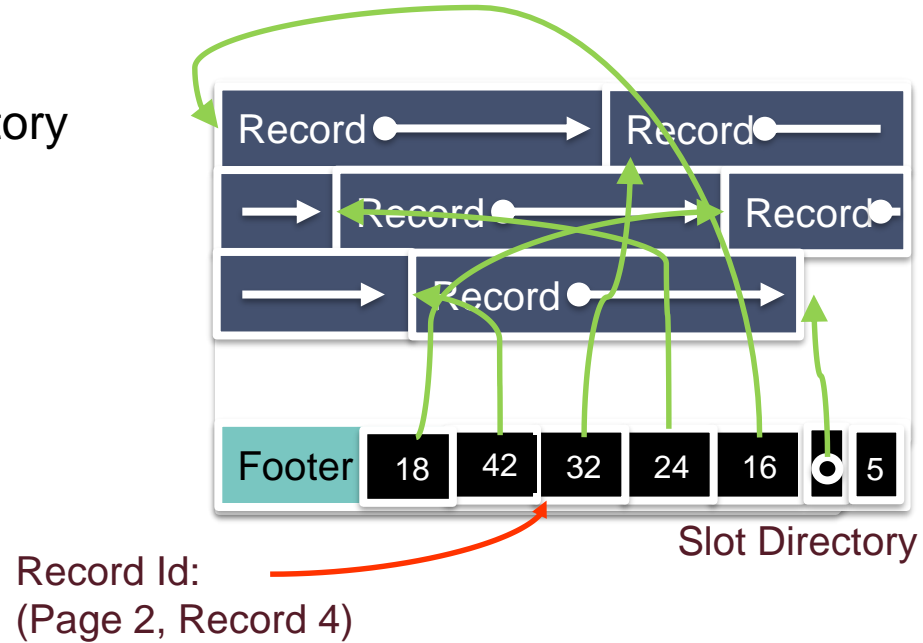
Slotted Page: Leading Questions

- Reorganize data on page
 - Is this safe?
 - Yes this is safe because records ids don't change.
- When should I reorganize?
 - We could re-organize on delete
 - Or wait until fragmentation blocks record addition and then reorganize.
 - Often pays to be a little sloppy if page never gets more records.
- What if we need more slots?
 - Let's see...



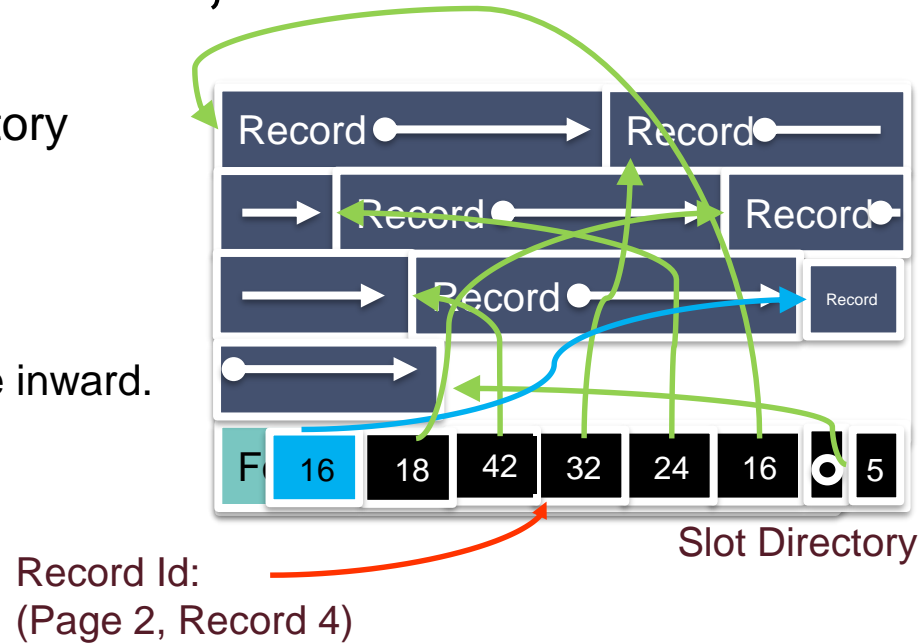
Slotted Page: Growing Slots

- Tracking number of slots in slot directory
 - Empty or full



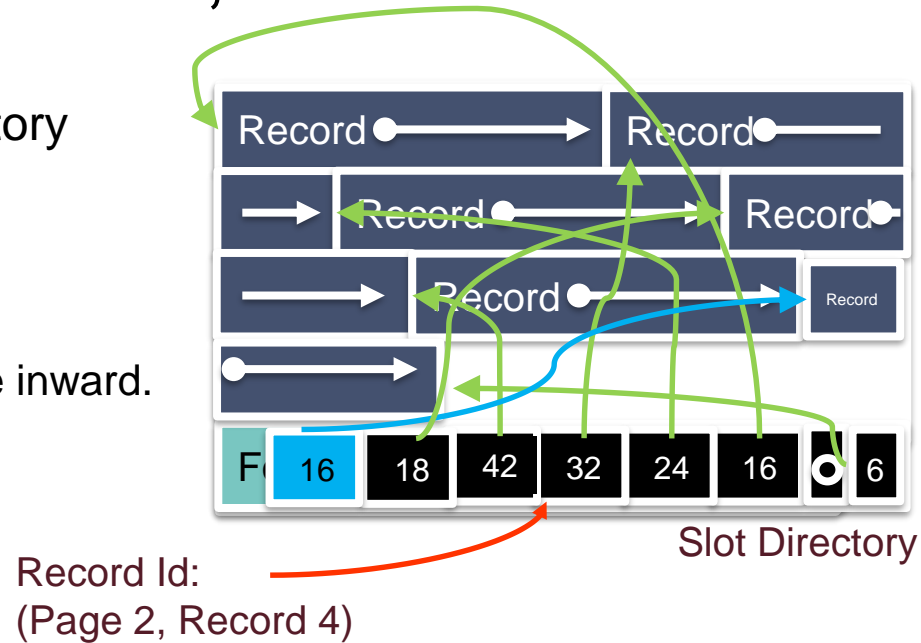
Slotted Page: Growing Slots, Pt. 2

- Tracking number of slots in slot directory
 - Empty or full
- Extend slot directory
 - Slots grow from end of page inward
 - Records grow from beginning of page inward.
 - Easy!



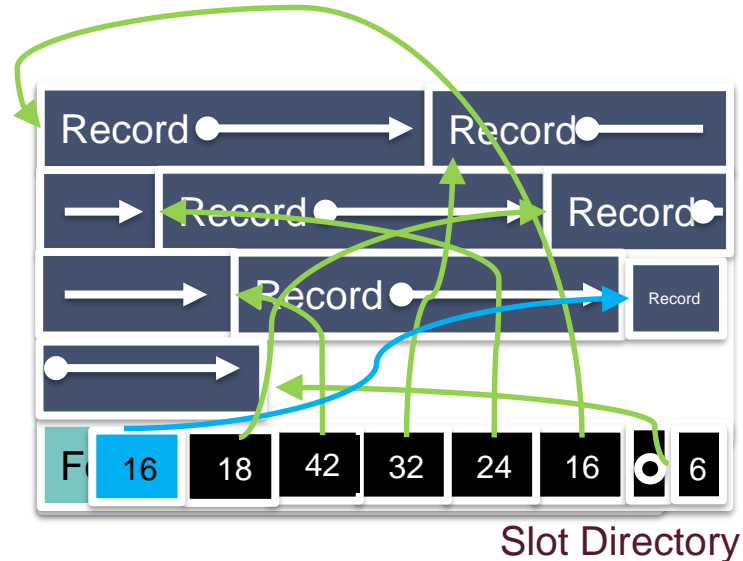
Slotted Page: Growing Slots, Pt. 3

- Tracking number of slots in slot directory
 - Empty or full
- Extend slot directory
 - Slots grow from end of page inward
 - Records grow from beginning of page inward.
 - Easy!
- And update count



Slotted Page: Summary

- Typically use Slotted Page
 - Good for variable and fixed length records
- Not bad for fixed length records too.
 - Why?
 - Re-arrange (e.g., sort) and squash null fields
 - But for a whole table of fixed-length non-null records, can be worth the optimization of fixed-length format



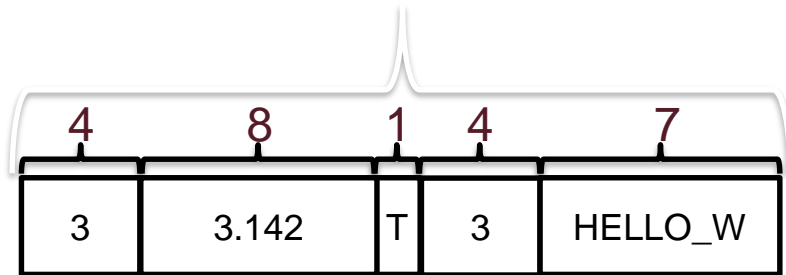
RECORD LAYOUT

Record Formats

- Relational Model →
 - Each record in table has some fixed type
- Assume System Catalog stores the Schema
 - No need to store type information with records (save space!)
 - Catalog is just another table ...
- Goals:
 - Records should be compact in memory & disk format
 - Fast access to fields (why?)
- Easy Case: Fixed Length Fields
- Interesting Case: Variable Length Fields

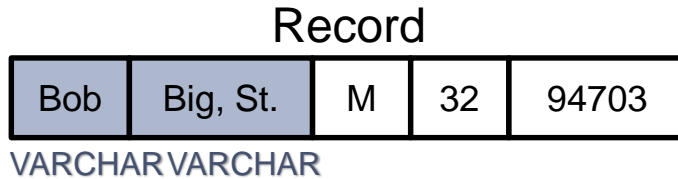
Record Formats: Fixed Length

- Field types same for all records in a file.
 - Type info stored separately in system catalog
- On disk byte representation same as in memory
- Finding i'th field?
 - done via arithmetic (fast)
- Compact? (Nulls?)



Record Formats: Variable Length

- What happens if fields are variable length?

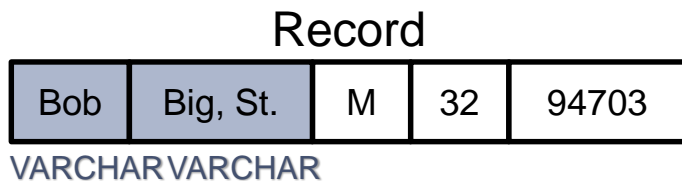


- Could store with padding? (Fixed Length)



Record Formats: Variable Length, Pt 2.

- What happens if fields are variable length?



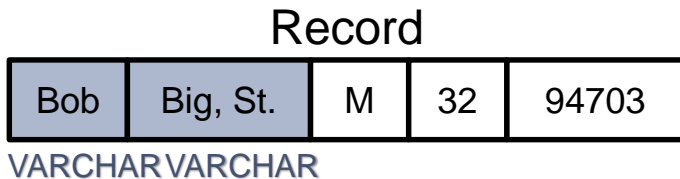
- Could use delimiters (i.e., CSV):



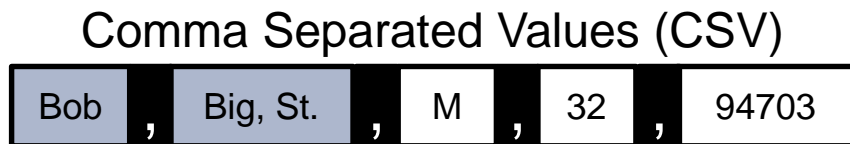
- Issues?

Record Formats: Variable Length, Pt. 3

- What happens if fields are variable length?



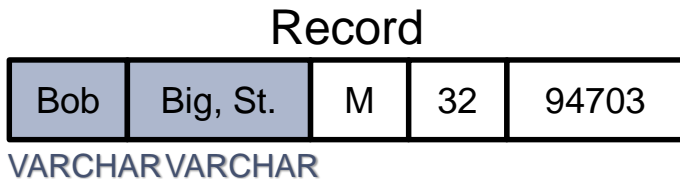
- Could use delimiters (i.e., CSV):



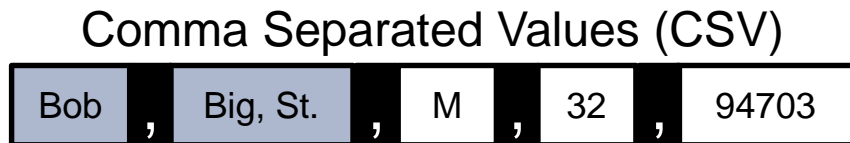
- Requires scan to access field
- What if text contains commas?

Record Formats: Variable Length, Pt 4.

- What happens if fields are variable length?



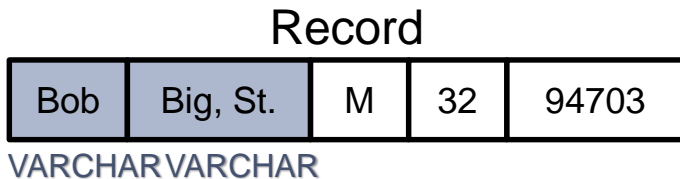
- Store length information before fields:



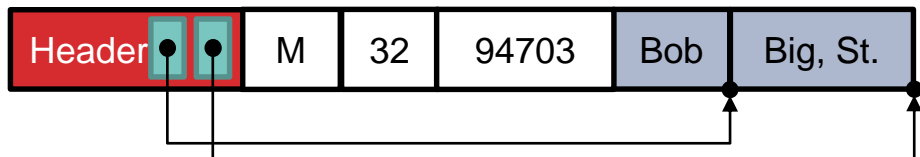
- Requires scan to access field
- Idea: Move all variable length fields to end enable fast access

Record Formats: Variable Length, Pt. 5

- What happens if fields are variable length?



- Introduce a record header



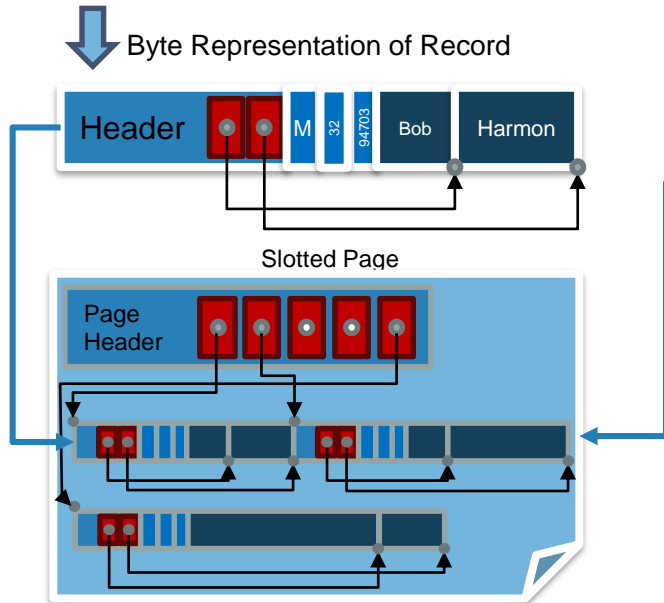
- Direct access & no “escaping”, other advantages?
 - Handle null fields easily →
 - useful for fixed length records too!

Summary 2

Record

Bob	Harmon	M	32	400
Varchar	Varchar	Char	Int	Int

SSN	Last Name	First Name	Age	Salary
123	Adams	Elmo	31	\$400
443	Grouch	Oscar	32	\$300
244	Oz	Bert	55	\$140
134	Sanders	Ernie	55	\$400



File

