THE UNIVERSITY of EDINBURGH

Advanced Database Systems

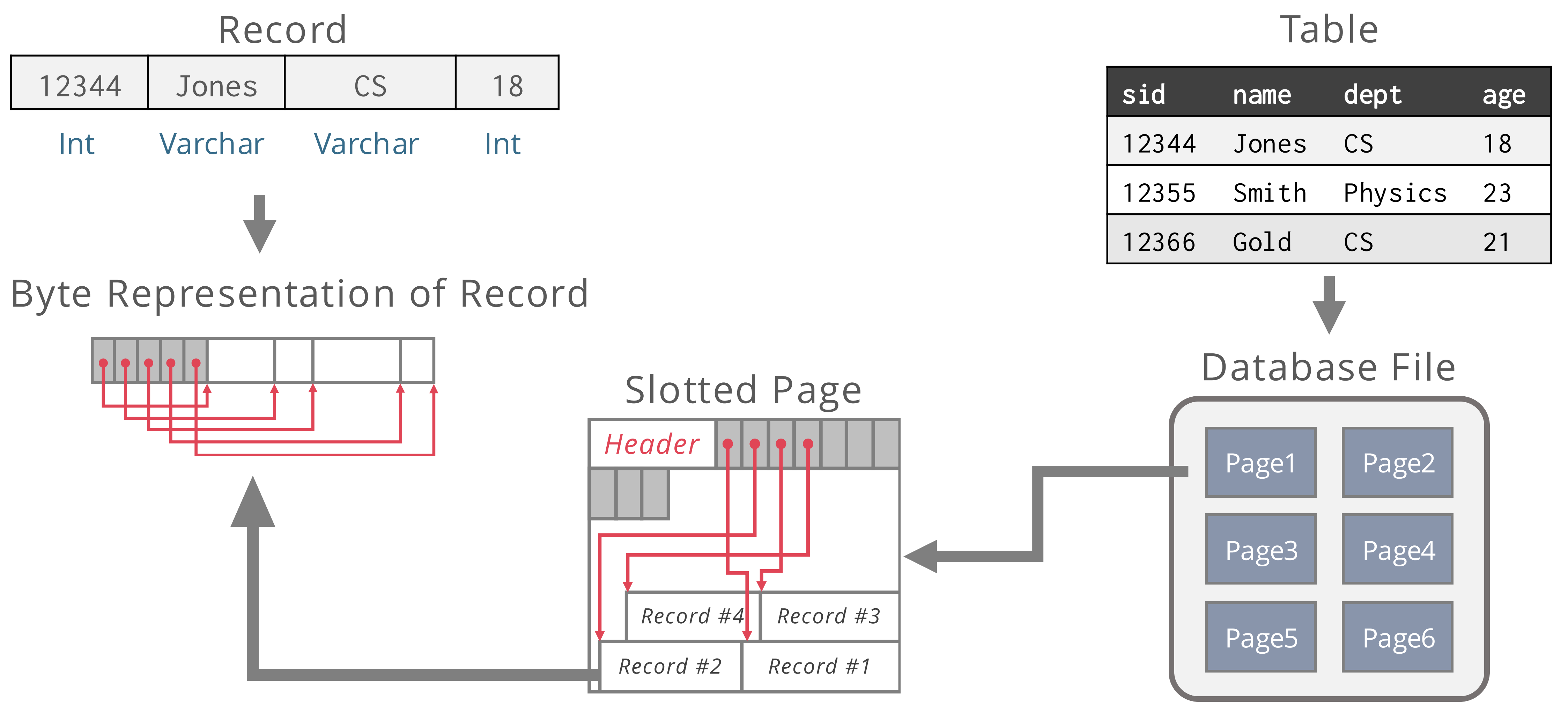
Spring 2025

Lecture #06:

Files, Pages, Records

R&G: Chapters 9.5-9.7

Overview: Representations



sid	name	dept	age
12344	Jones	CS	18
12355	Smith	Physics	23
12366	Gold	CS	21

Outline

Storage Media

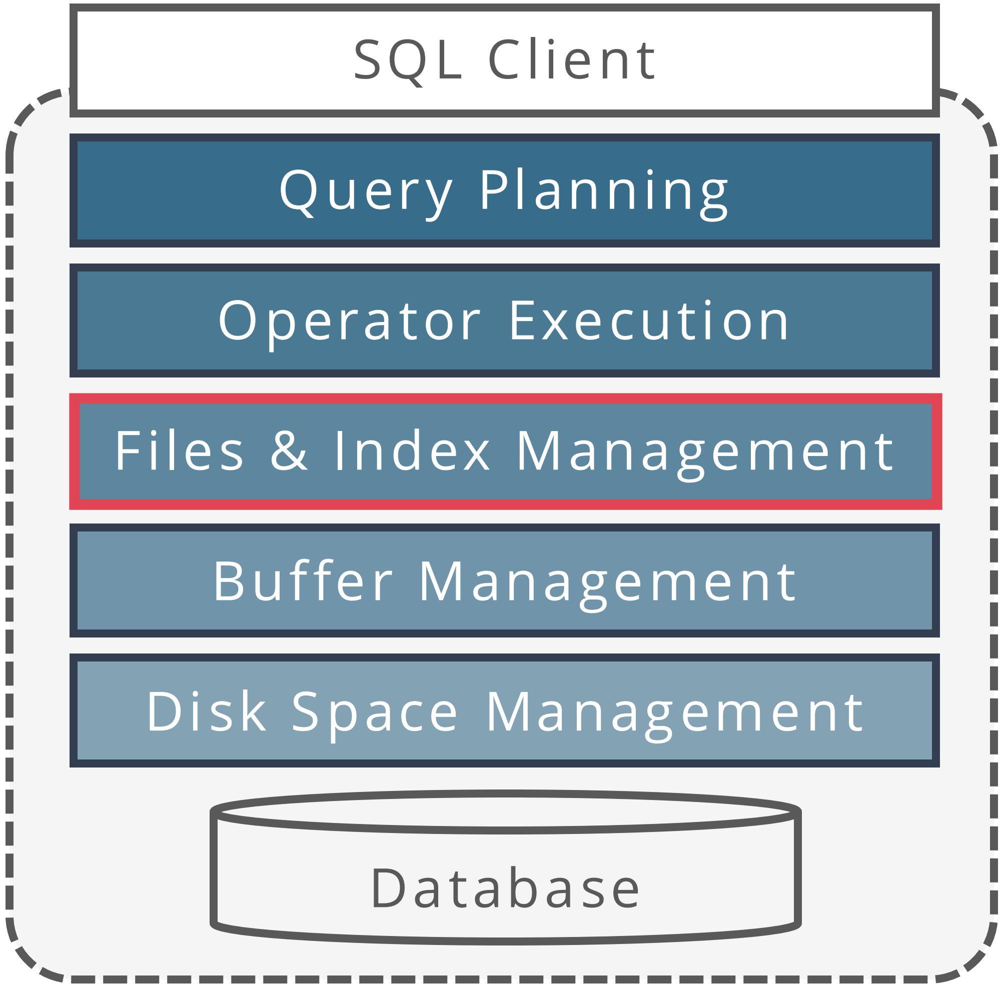
Disk Space Management

Buffer Management

File Layout

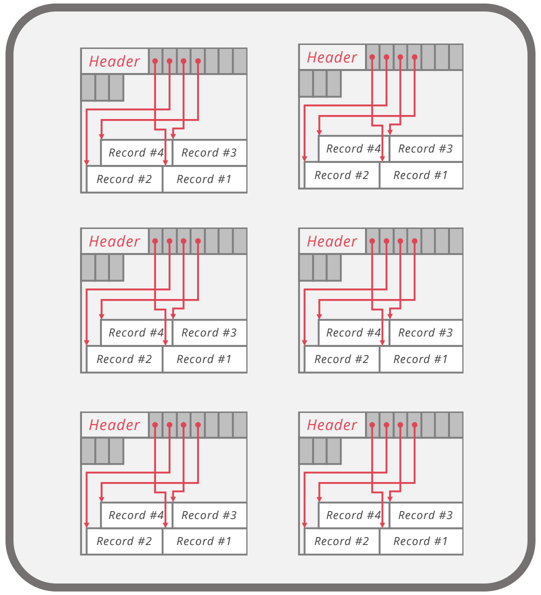
Page Layout

Record Layout



Files of Pages of Records

- Tables stored as logical (database) files
  - A file consists of one or more pages
  - Each page contains one or more 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
- Page management is oblivious to their actual content



Database File

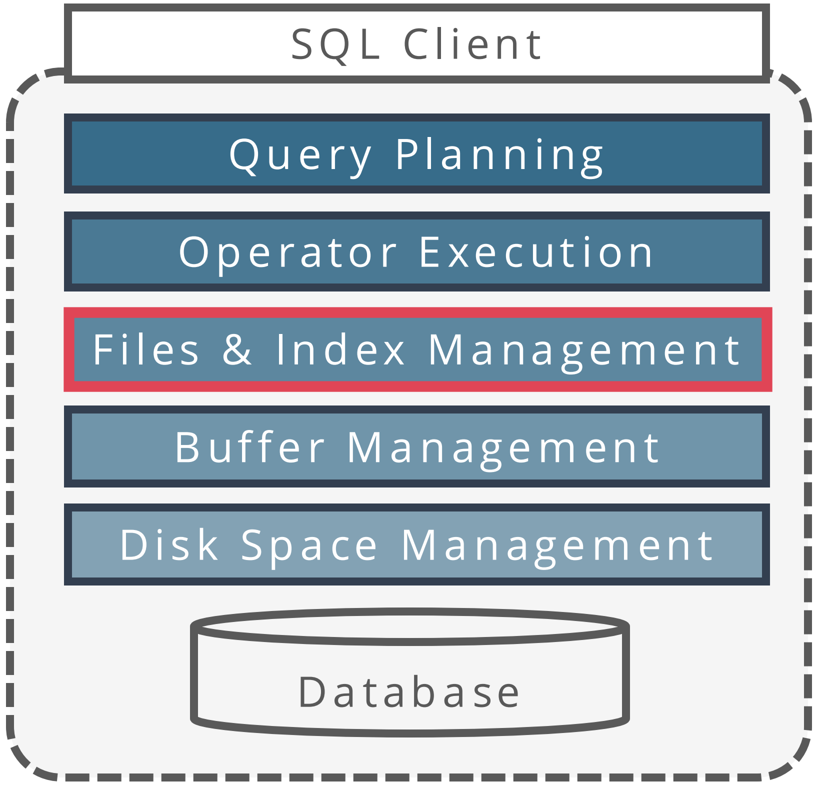
Files of Pages of Records

Database file: A collection of pages, each containing a collection of records

- Could span multiple OS files and even machines

API for higher levels of the DBMS:

- Create/delete a file
- Insert/delete/modify a record
- Fetch a particular record by **record ID**
  - Record ID = (page ID, location on page)
- Scan all records
  - possibly with a predicate on the desirable records



DB File Organisation

- Method of arranging a file of records
  - At this point in the hierarchy, we do not care what is page format
- Different types exist, each ideal for some situations & not so good in others:
  - Heap Files
    - Records placed arbitrarily across pages
  - Sorted Files
    - Pages and records are in sorted order
  - Index Files
    - B+ trees, hash-based files
    - May contain records or point to records in other files

Heap File

Most important type of files in a database

Collection of records in **no particular order**

- Not to be confused with “heap” data-structure

As file shrinks/grows, pages allocated/deallocated

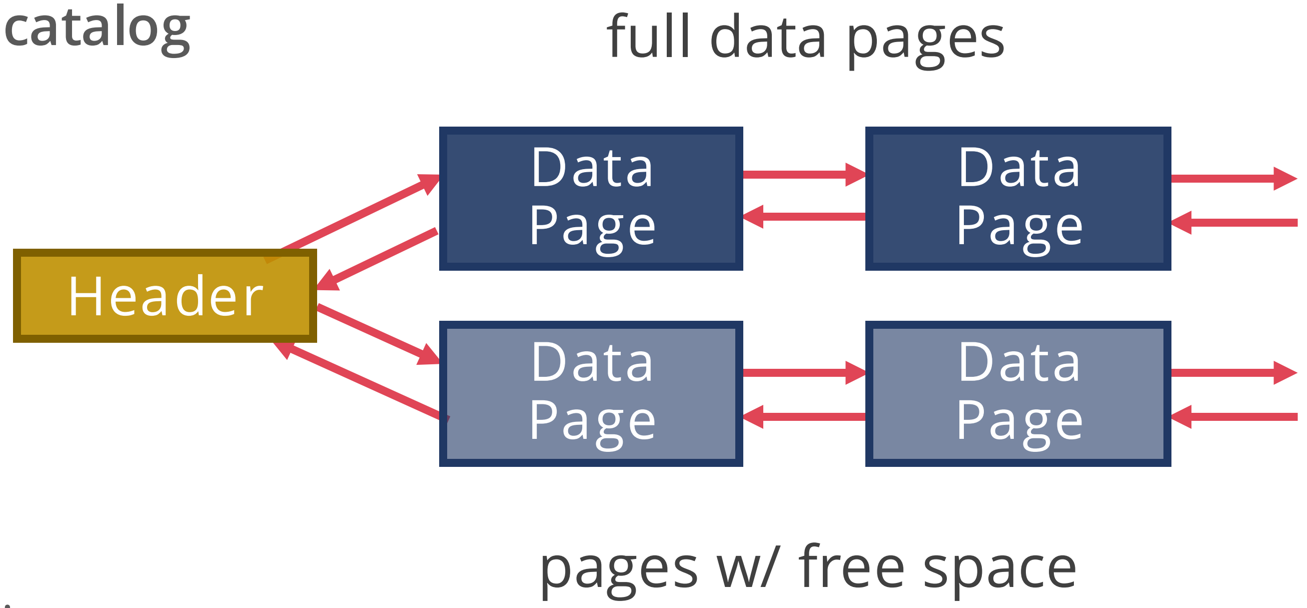
To support record level operations, we must

- 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: Linked List

Doubly linked lists of pages

- Header page allocated when the file is created
- Header page ID stored in the **system catalog**
- Initially both page lists are empty
- Each page keeps track of the free space in itself



Easy to implement, but

- Most pages end up in the free space list
- Finding a page with sufficient empty space may search many pages

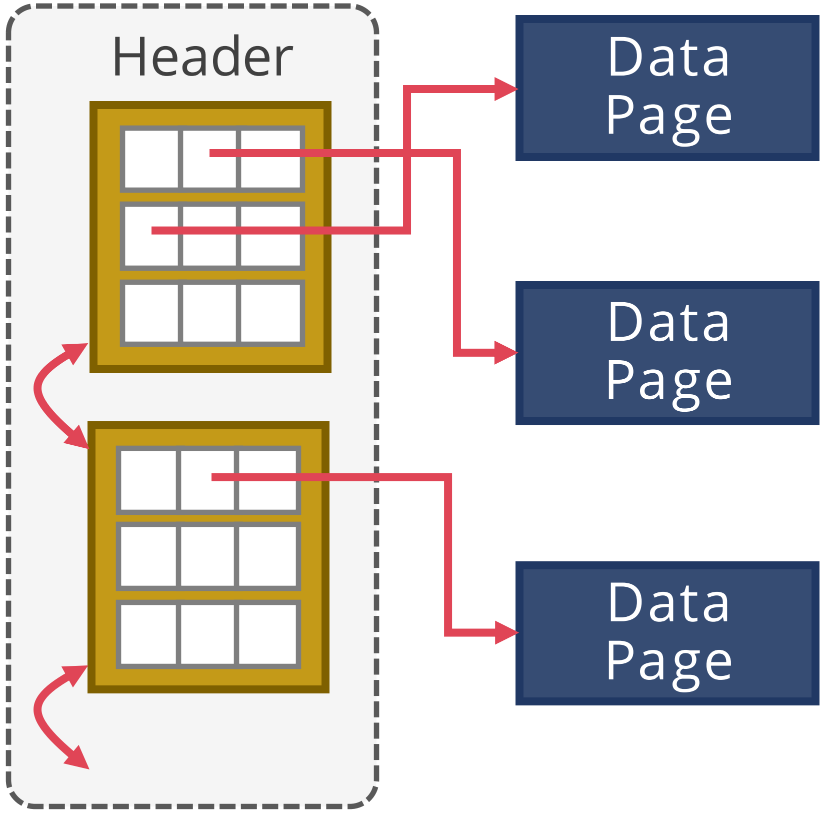
Heap File: Page Directory

Directory = set of special pages storing metadata about data pages

Each directory entry
- Identifies a data page & records **#free bytes** on it

Free space search more efficient
- Far fewer pages read to find a page to fit a record
  - One header page reveals free space of many pages
- Header pages accessed often ⇒ likely in cache

Small memory overhead to host the directory



Page Directory

Outline

Storage Media

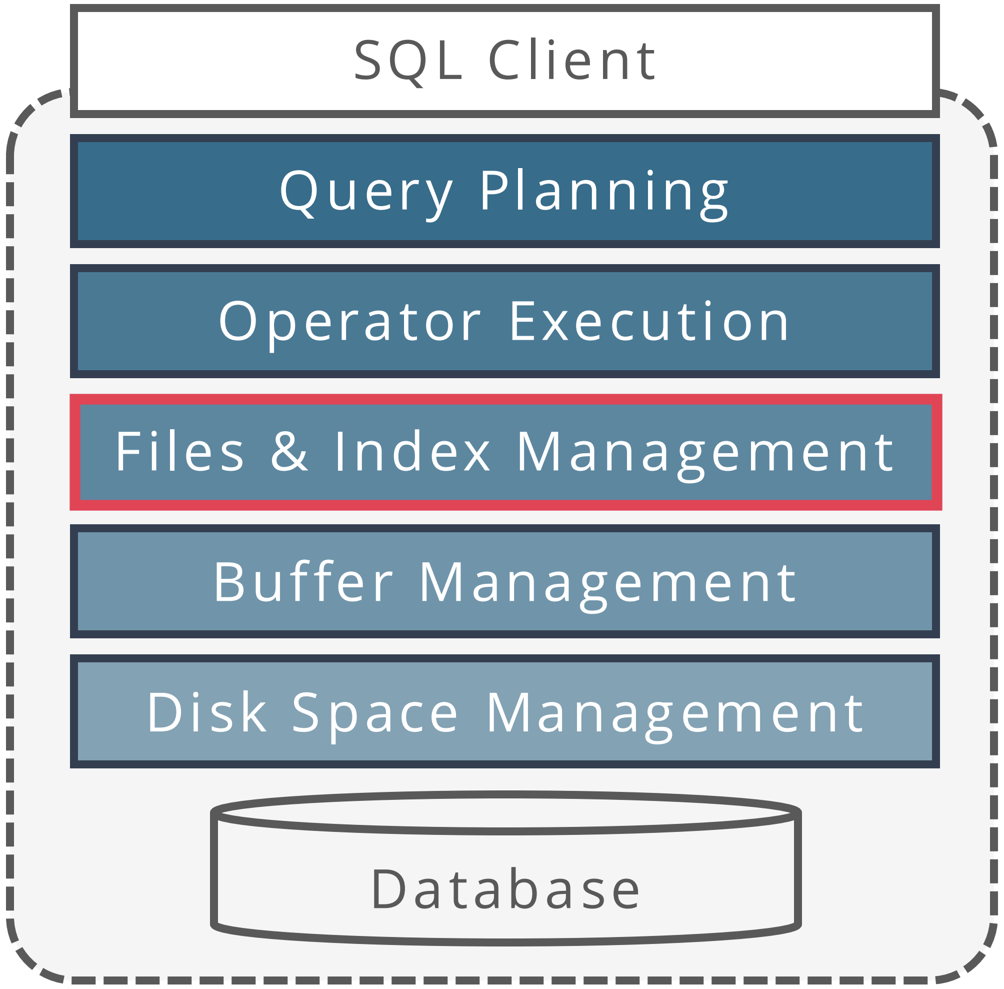
Disk Space Management

Buffer Management

File Layout

Page Layout

Record Layout



Page Layout

How to organize the data stored inside the page

Header stores metadata about the page

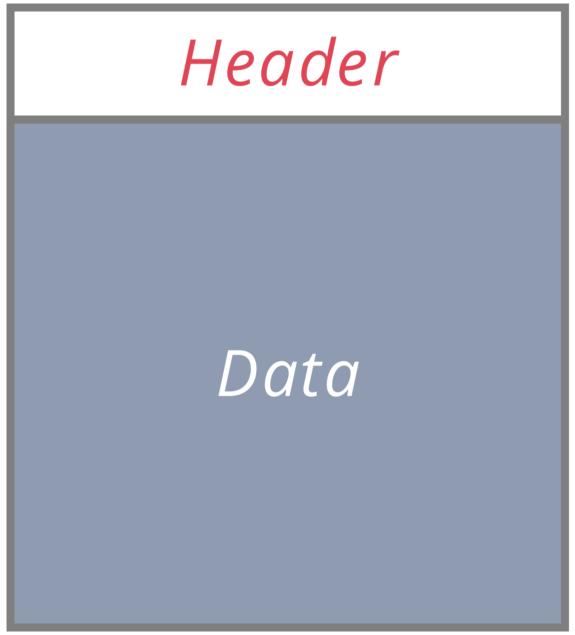
- page ID, #records, free space, next/prev pointers, etc.

Find record by record ID, insert, delete records

- **record ID** (rid) = (page ID, offset in page)

Things to consider:

- Record length? Fixed or variable
- Page packing? Packed or unpacked



Page

Fixed-Length Records

Records are made up of multiple **fields**

- Fields = values for columns in a table

We have fixed-length records when field lengths are consistent

- The first field always has N bytes, the second field always has M bytes, etc.

⇒ Record lengths are fixed

- Every record is always the same number of bytes
- Notice that the implication might not be true the other way

We can store fixed-length records in two ways: packed and unpacked

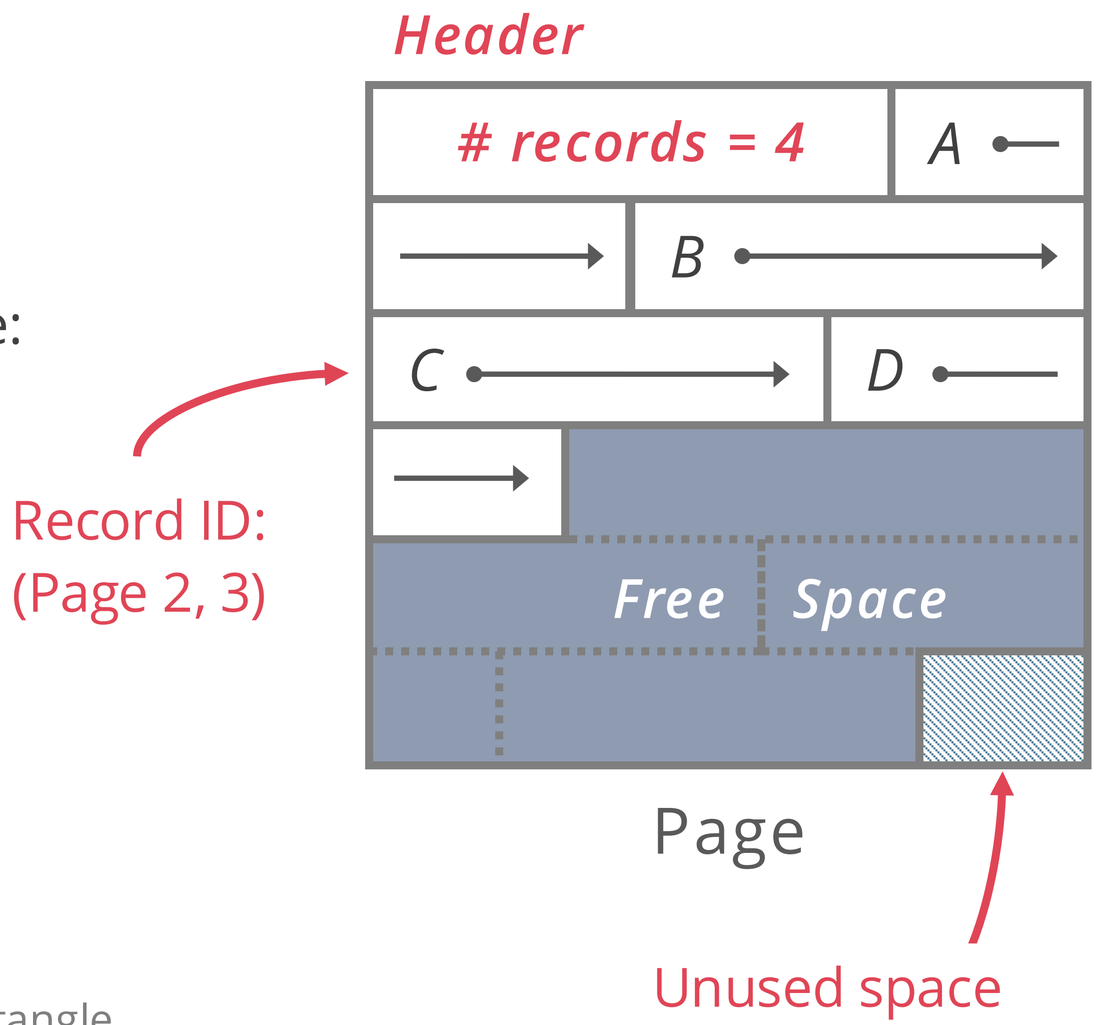
Fixed-Length Records, Packed

No gaps between records

Record ID = (page ID, position in page)

Easy to compute the offset of a record in the page:

sizeof(Header) + (position – 1) * *sizeof*(Record)



Note:

Data is always stored in linear order

For presentation, we “wrap around” the linear order into a rectangle

Fixed-Length Records, Packed

No gaps between records

Record ID = (page ID, position in page)

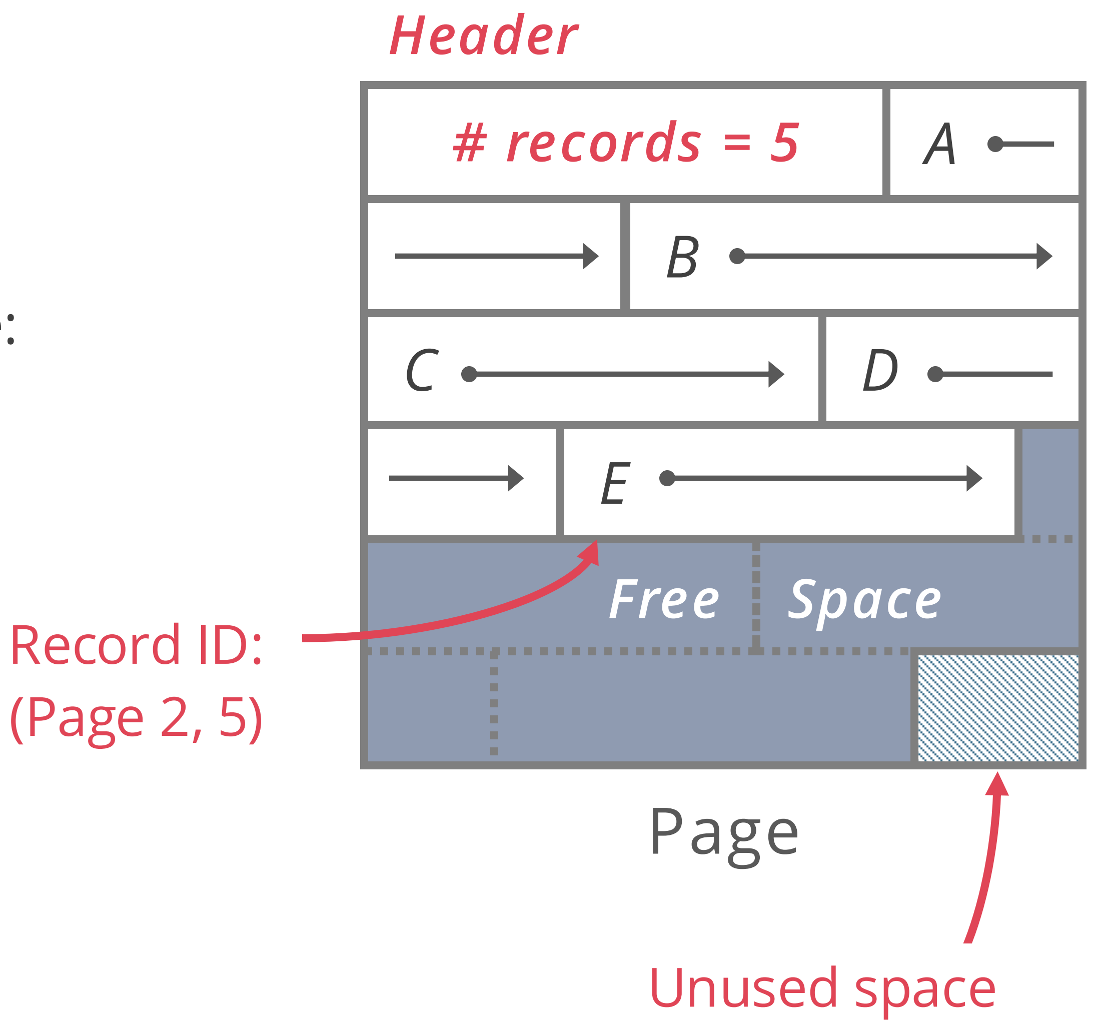
Easy to compute the offset of a record in the page:

sizeof(Header) + (position – 1) * *sizeof*(Record)

Insert record

Just append a new record to the end

sizeof(Header) + # records * *sizeof*(Record)



Fixed-Length Records, Packed

No gaps between records

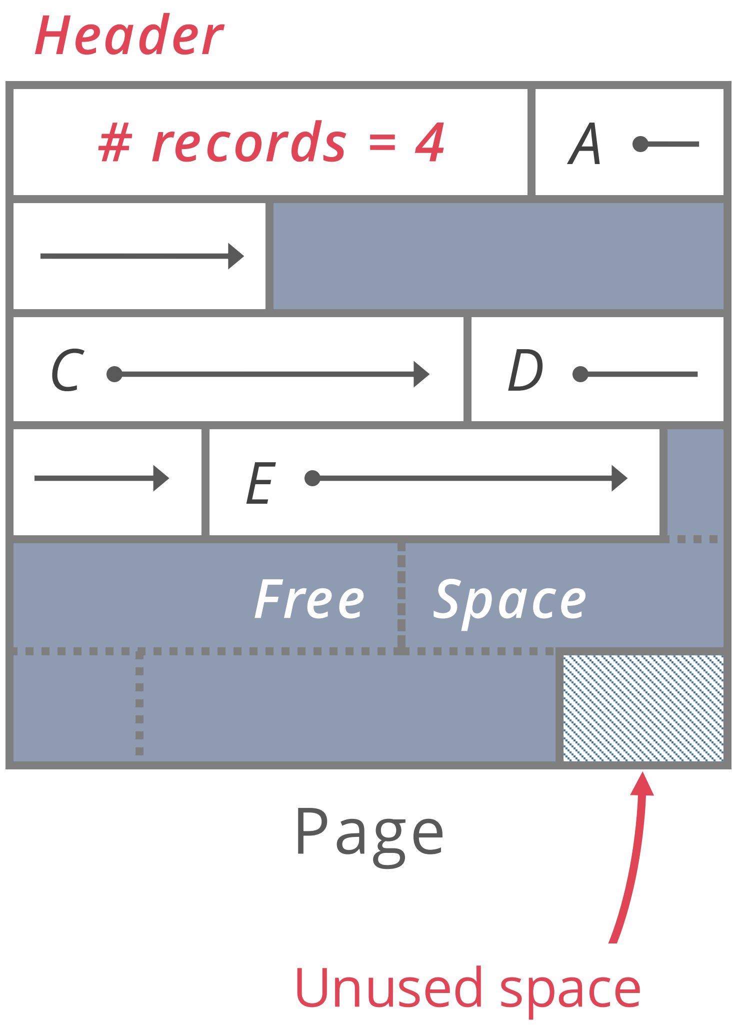
Record ID = (page ID, position in page)

Easy to compute the offset of a record in the page:

- *sizeof*(Header) + (position – 1) * *sizeof*(Record)

Delete record

- Move the last record to the emptied slot
- But this changes the last record's ID!
  - Not always desirable
  - Updating record ID pointers could be expensive if they're in other files



Fixed-Length Records, Packed

No gaps between records

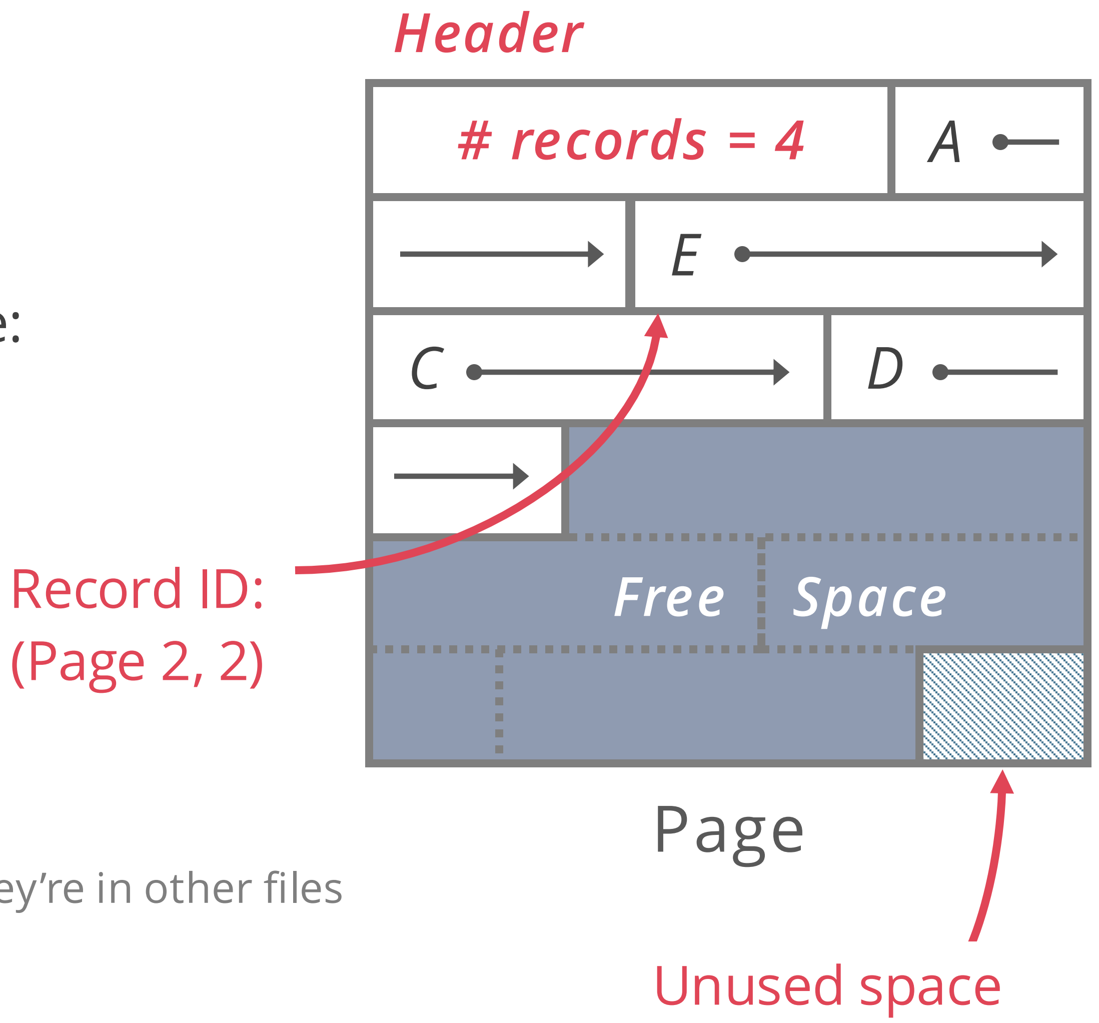
Record ID = (page ID, position in page)

Easy to compute the offset of a record in the page:

sizeof(Header) + (position – 1) * *sizeof*(Record)

Delete record

- Move the last record to the emptied slot
- But this changes the last record's ID!
  - Not always desirable
  - Updating record ID pointers could be expensive if they're in other files



FIXED-LENGTH RECORDS, UNPACKED (BITMAP)

Allow gaps between records

Record ID = (page ID, position in page)

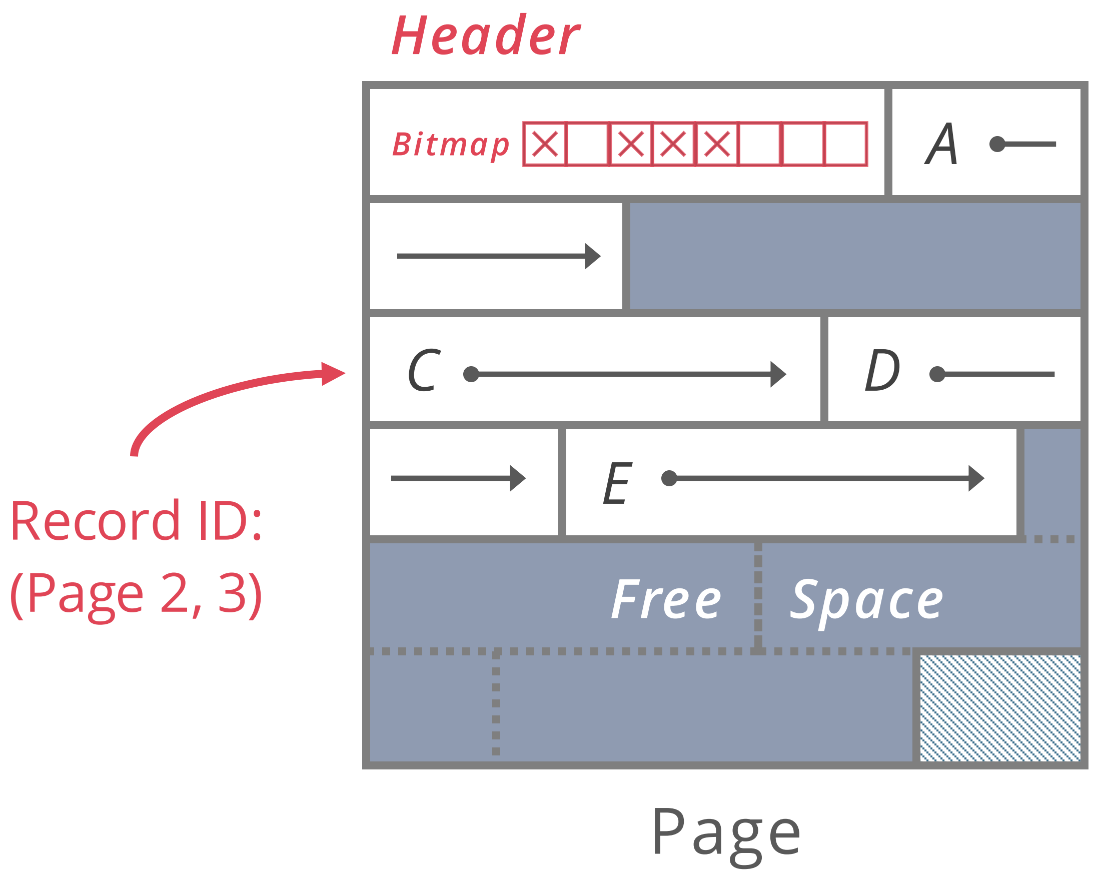
Use a **bitmap** to keep track of where the gaps are

Insert record

- Find first empty slot by scanning the bitmap

Delete record

- Clear bit in the bitmap



Fixed-Length Records, Unpacked (Free List)

Alternative to using bitmap

Link all free slots into a free list

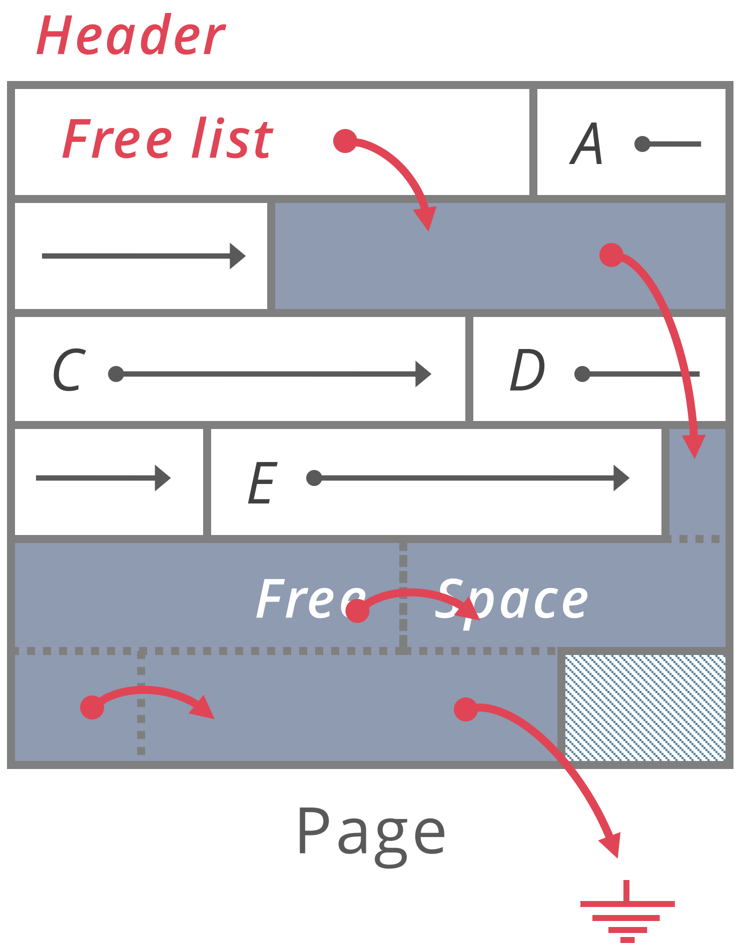
- Each link points to the beginning of a free slot, last is null

Insert record

- Insert into slot pointed by head of free list
- Set next free slot as new head

Delete record

- Set slot of deleted record as new head



Fixed-Length Records, Unpacked (Free List)

Alternative to using bitmap

Link all free slots into a free list

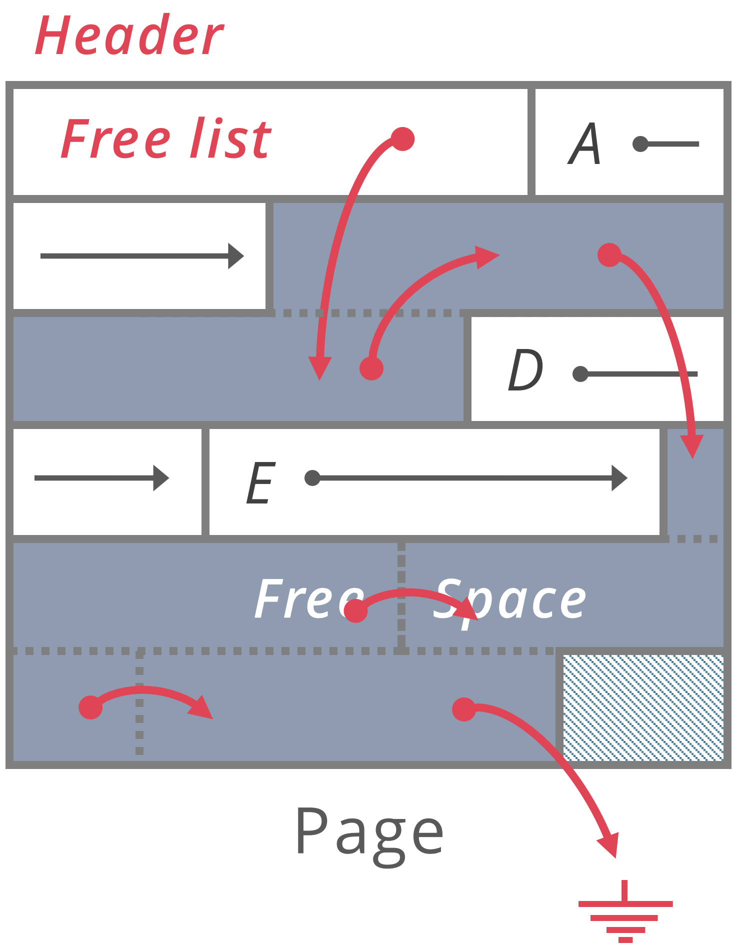
- Each link points to the beginning of a free slot, last is null

Insert record

- Insert into slot pointed by head of free list
- Set next free slot as new head

Delete record

- Set slot of deleted record as new head
- Example: after deleting record C



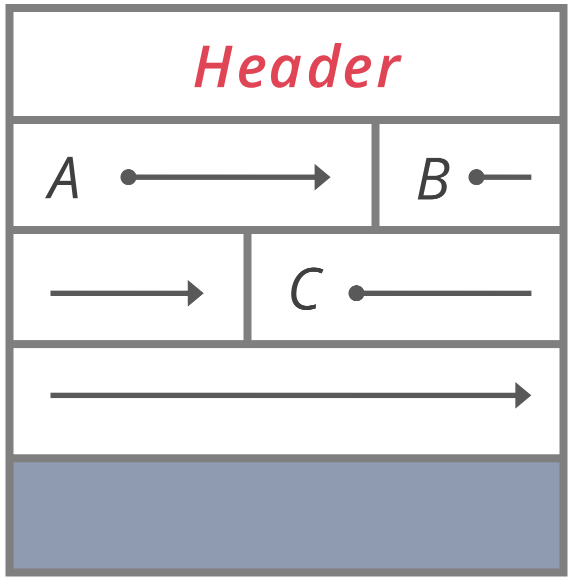
Variable-Length Records

Variable-length records may not have field lengths consistent

E.g.: The third field may take 0 to 4 bytes

How do we know where each record begins?

What happens when we add and delete records?



Page

Slotted Pages

Most common layout scheme is called **slotted pages**

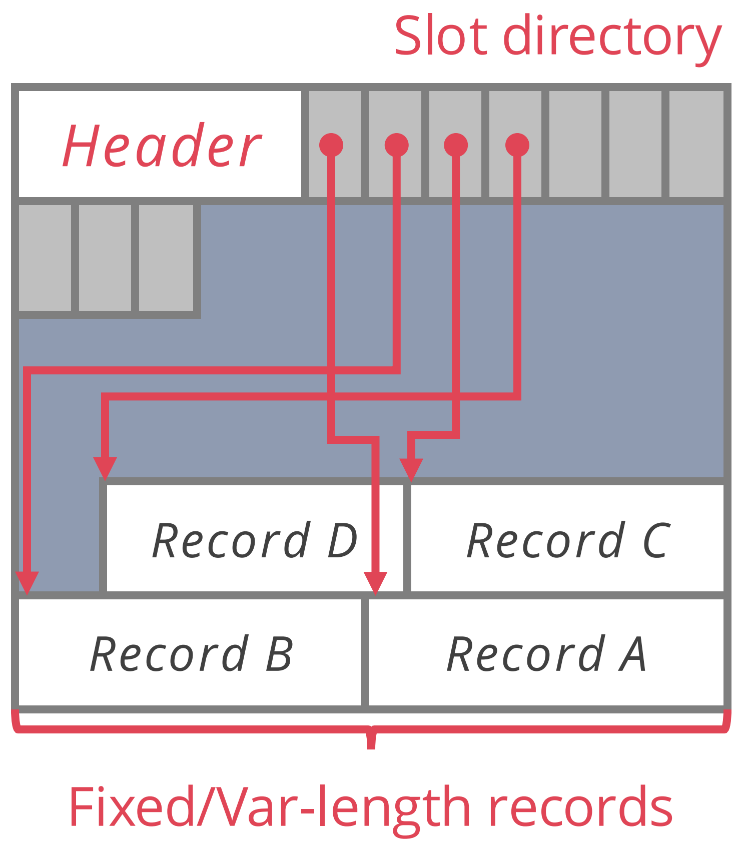
Slot directory maps "slots" to the records' starting position offsets

- Record ID = (page ID, slot ID)

Header keeps track of:

- The number of used slots
- The offset of the last slot used

Records stored at the end of page



Slotted Pages

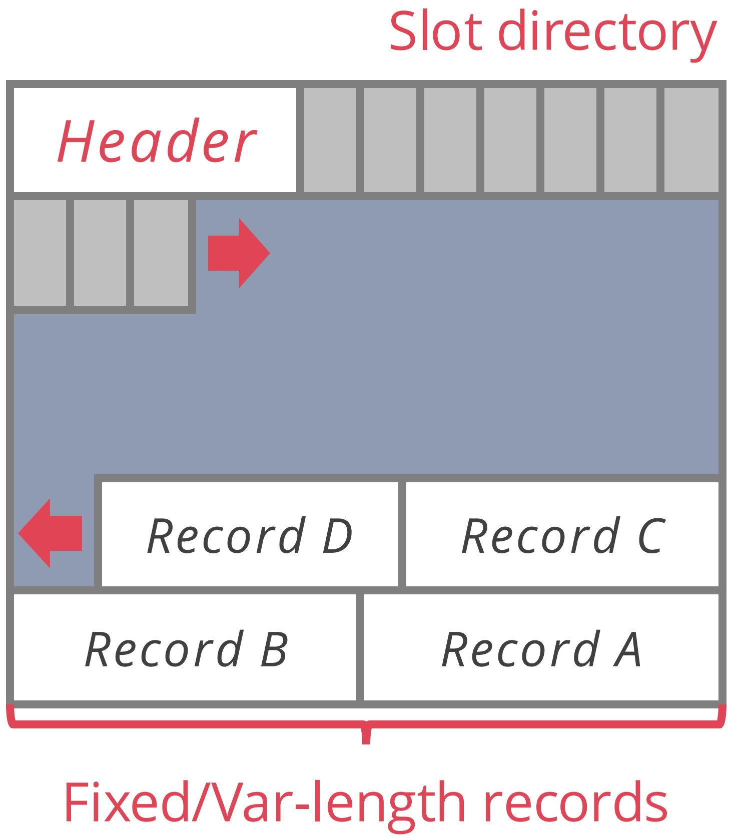
Records can be moved without changing rid

Delete record

- Set slot offset to -1, delete slot only if last
- Move records to fill up the whole or defragment space periodically

Insert record

- Find a slot with offset -1 or create if none
- Allocate just the right amount of space
- Defragment if not enough free space



Outline

Storage Media

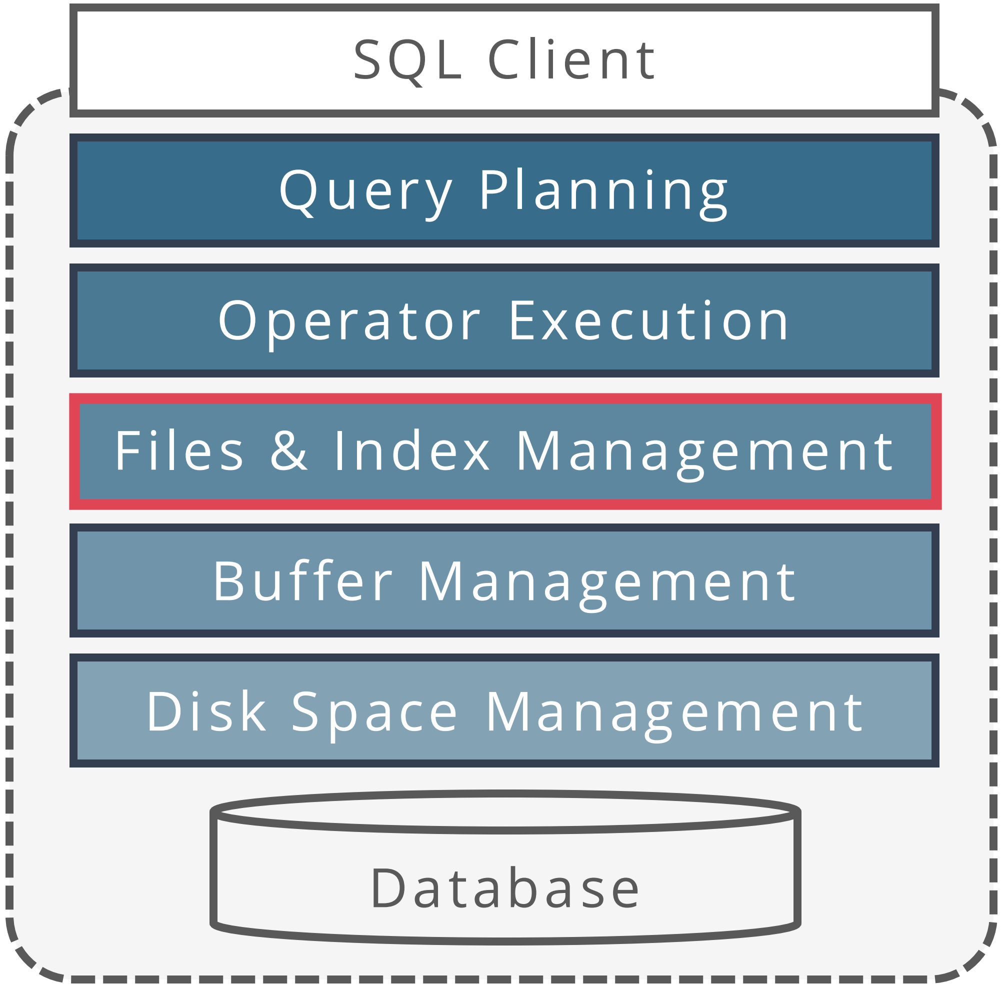
Disk Space Management

Buffer Management

File Layout

Page Layout

Record Layout



Record Layout

Relational model

- Each record in table has some fixed type

We do **not** need to store metadata about the schema

- Information about field types is stored in the **system catalog**
- System catalog is just another set of tables

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 Layout: Fixed-Length Records

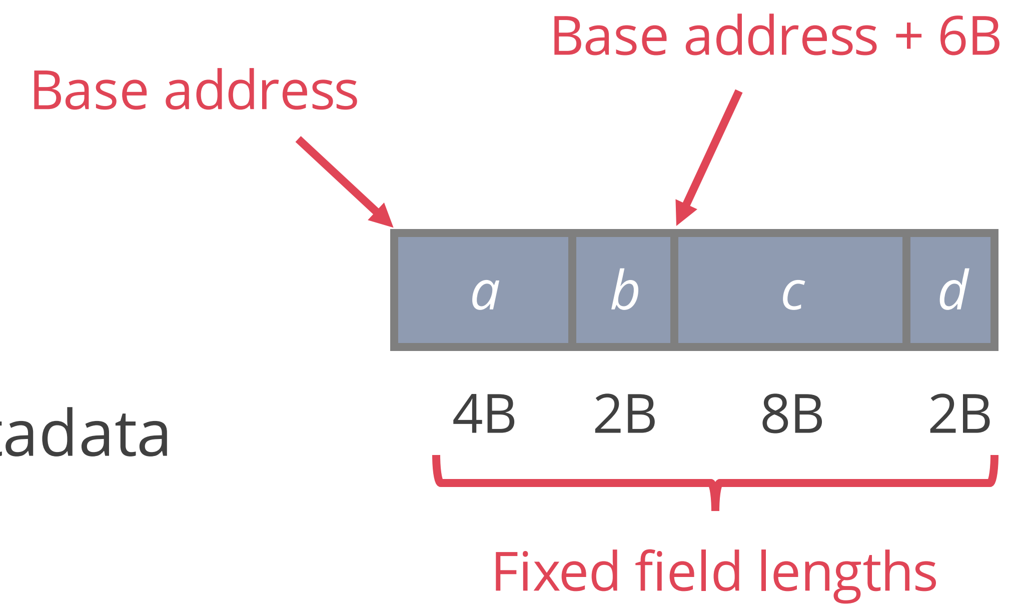
Each field has a **fixed** length

Direct access to record fields

- Done via arithmetic (fast)

Each record can have a header storing metadata

- E.g., bitmap for **NULL** values
- No need to store information about the schema

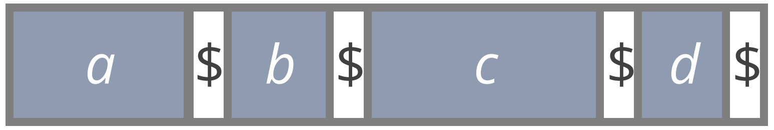


Variable-Length Records

Some fields have variable length

Two ways to store variable length records:

1. **Fields delimited by special symbols**
    - Access to fields requires a scan of the record
    - Special symbols in fields require “escaping”



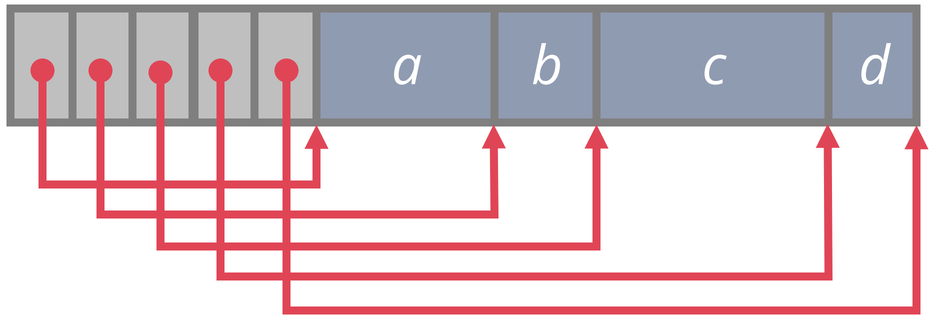
Variable-Length Records

2. Array of field offsets

Direct access to fields & no “escaping”

Useful for fixed-length records too

Clean way of dealing with NULL values



Aside: this is actually not sufficient for storing NULL values for all types

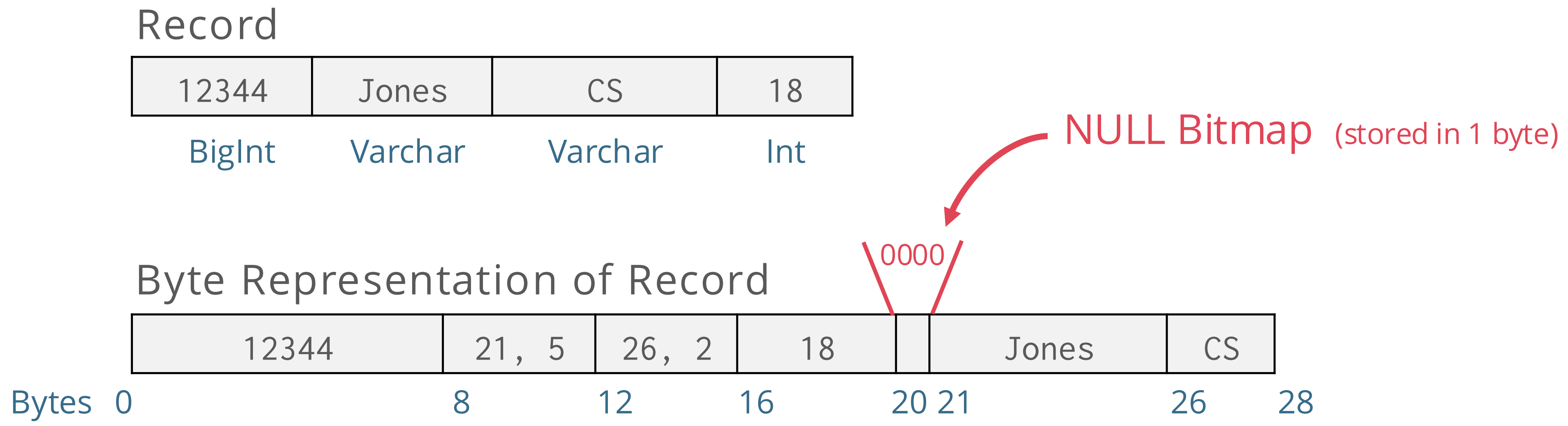
Cannot distinguish between empty string (“”) and NULL

Need some extra metadata (e.g. bitmap in record header or special char in field), which varies widely between different DBMSs

Variable-Length Records

Possible implementation

- Fields are stored in order
- Variable-length fields represented by fixed size **(offset, length)**, with actual data stored after all fixed-length fields
- NULL values represented by NULL-value bitmap



SUMMARY

DB file contains pages, and records within pages

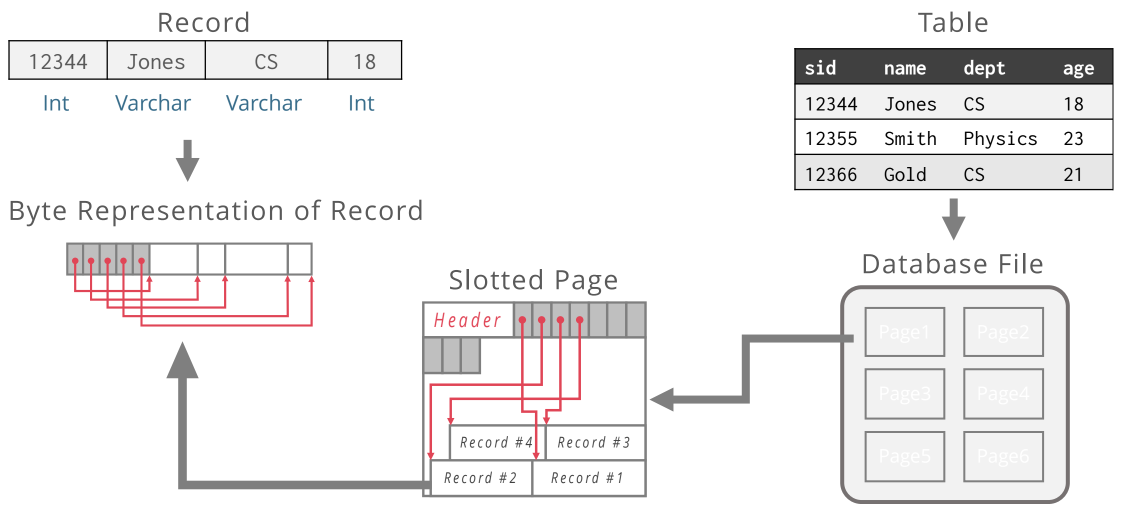
Heap files: unordered records organized with directories

Page layout

- Fixed-length packed and unpacked
- Variable length records in slotted pages

Variable-length record layout

- Direct access to i-th field and NULL values



sid	name	dept	age
12344	Jones	CS	18
12355	Smith	Physics	23
12366	Gold	CS	21