Advanced Database Systems

Spring 2024

Lecture 12:

Files, Pages, Records

R&G: Chapters 9.5-9.7

OVERVIEW: REPRESENTATIONS

Record

12344	Jones	CS	18
Int	Varchar	Varchar	Int
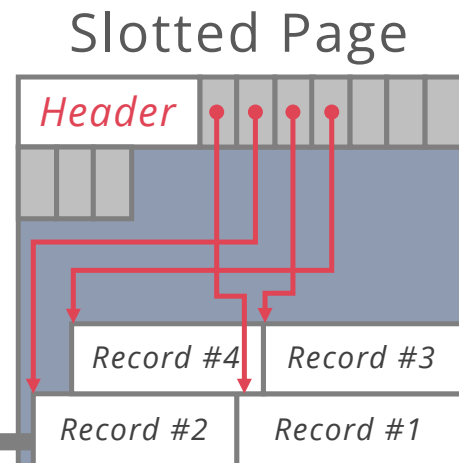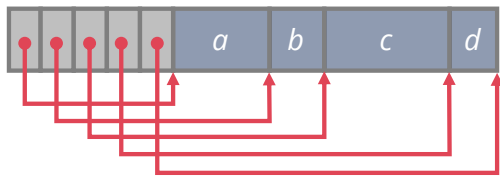
Byte Representation of Record

| | | | | | a | b | c | d |

Table

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

Database File

Page1	Page2
Page3	Page4
Page5	Page6

Slotted Page

Header

Record #4 | Record #3

Record #2 | Record #1

OUTLINE

Storage Media

Disk Space Management

Buffer Management

File Layout

Page Layout

Record Layout

| SQL Client |
| Query Planning |
| Operator Execution |
| Files & Index Management |
| Buffer Management |
| Disk Space Management |
| Database |
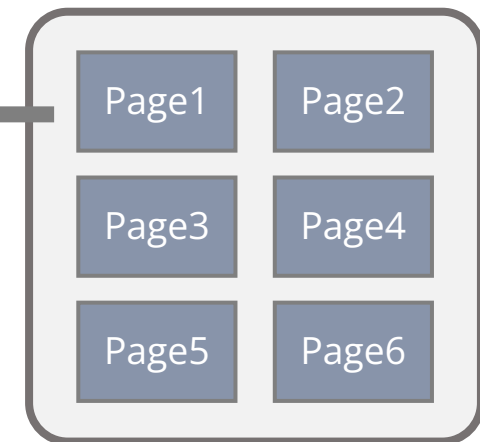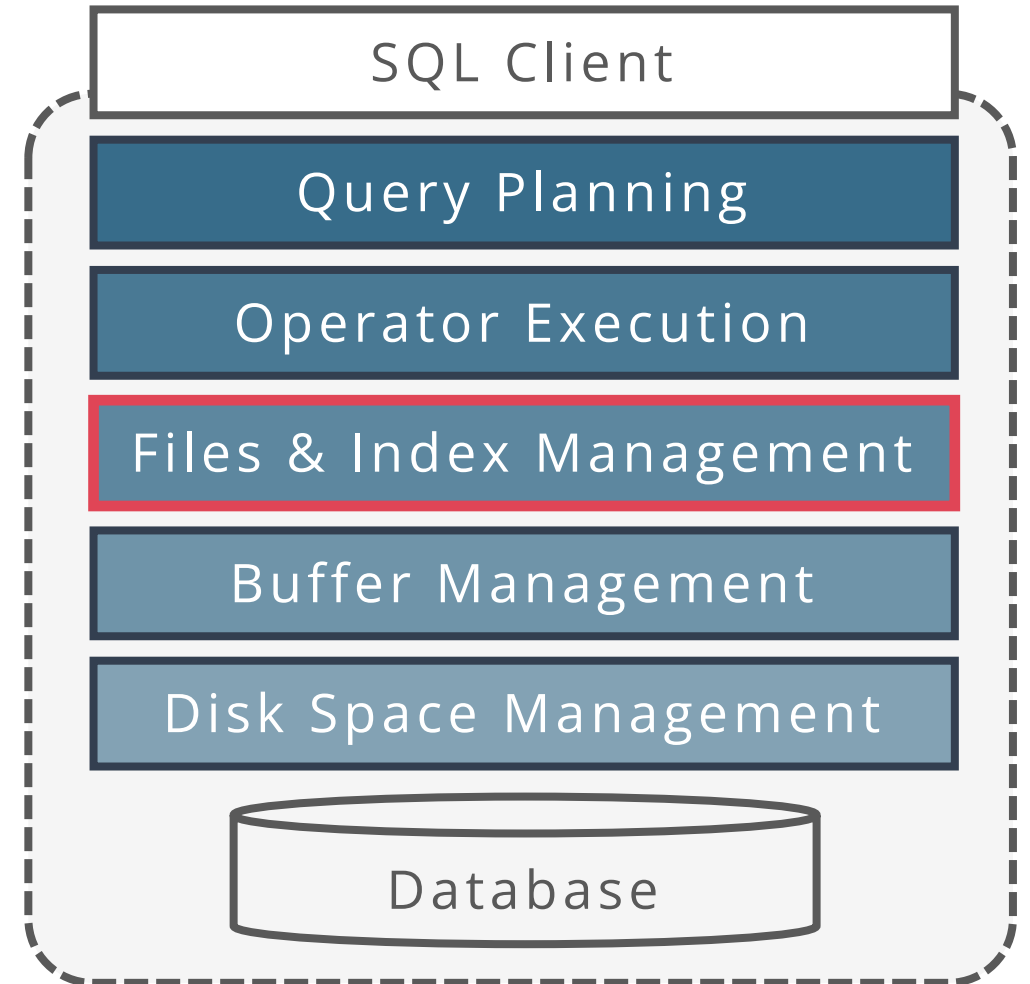
FILES OF PAGES OF RECORDS

Tables stored as **logical (database) files**

A file consists of one or more **pages**

Each page contains one or more **records**
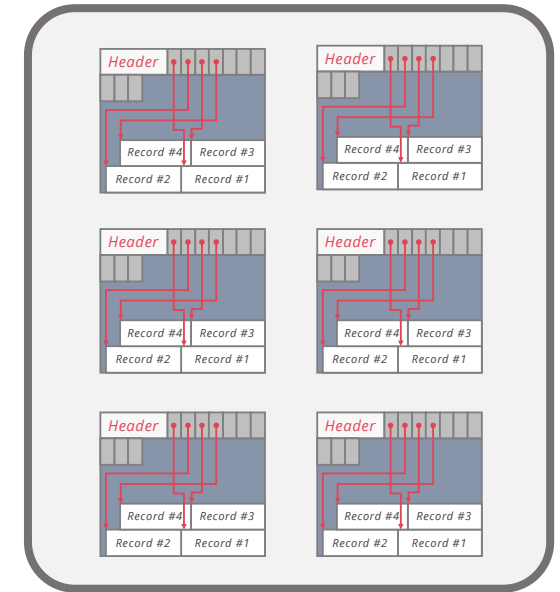
Pages are managed

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

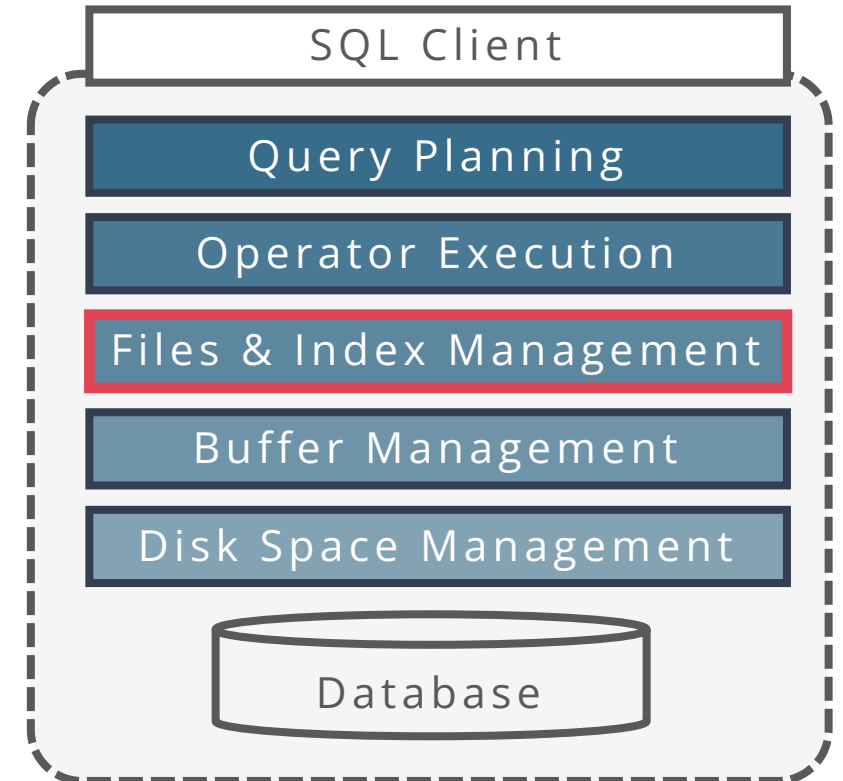API for higher levels of the DBMS:

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

SQL Client
Query Planning
Operator Execution
Files & Index Management
Buffer Management
Disk Space Management
Database

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
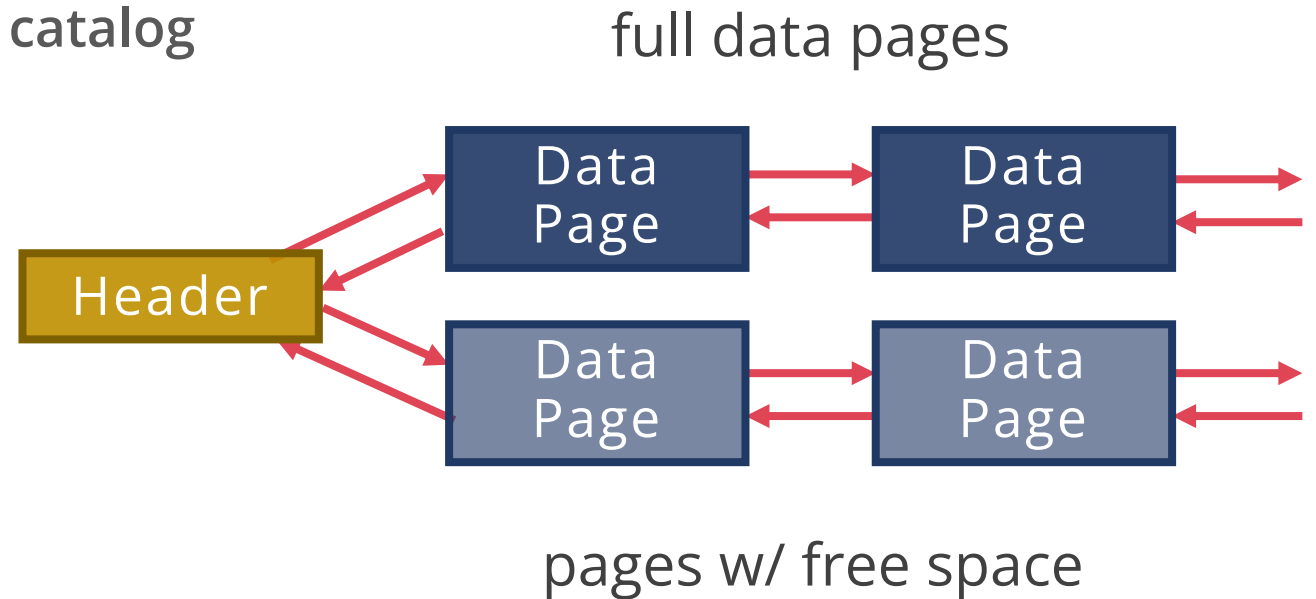
Heap File: Linked List

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

full data pages

pages w/ free space

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

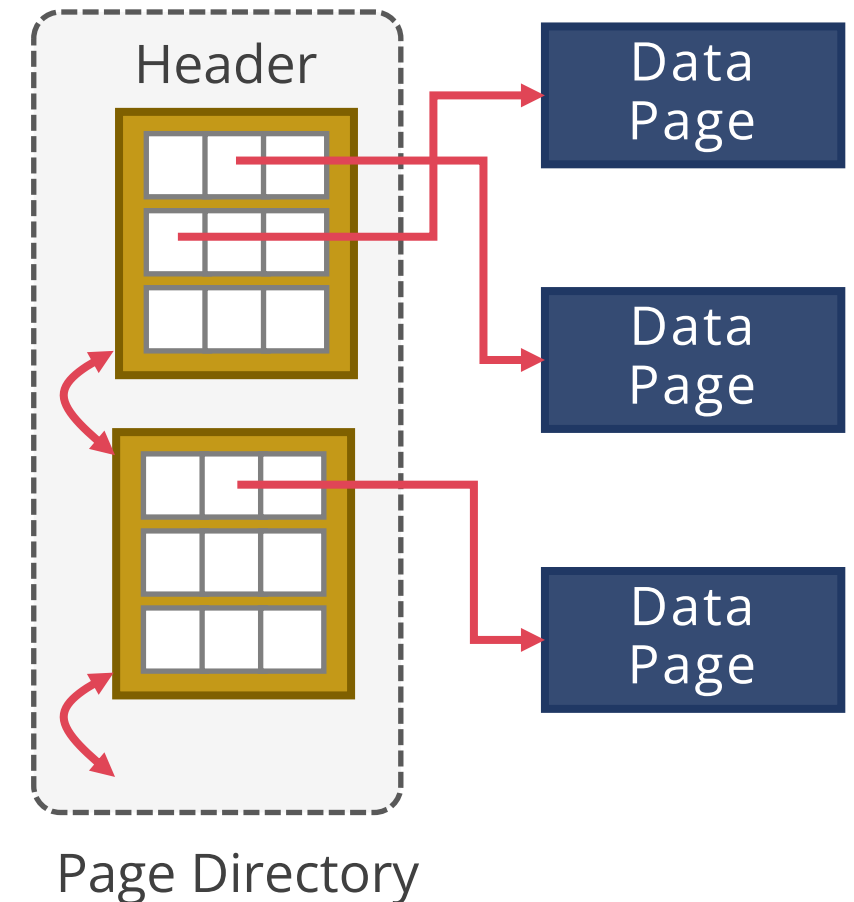Identifies a data page & records **#free bytes** on it

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
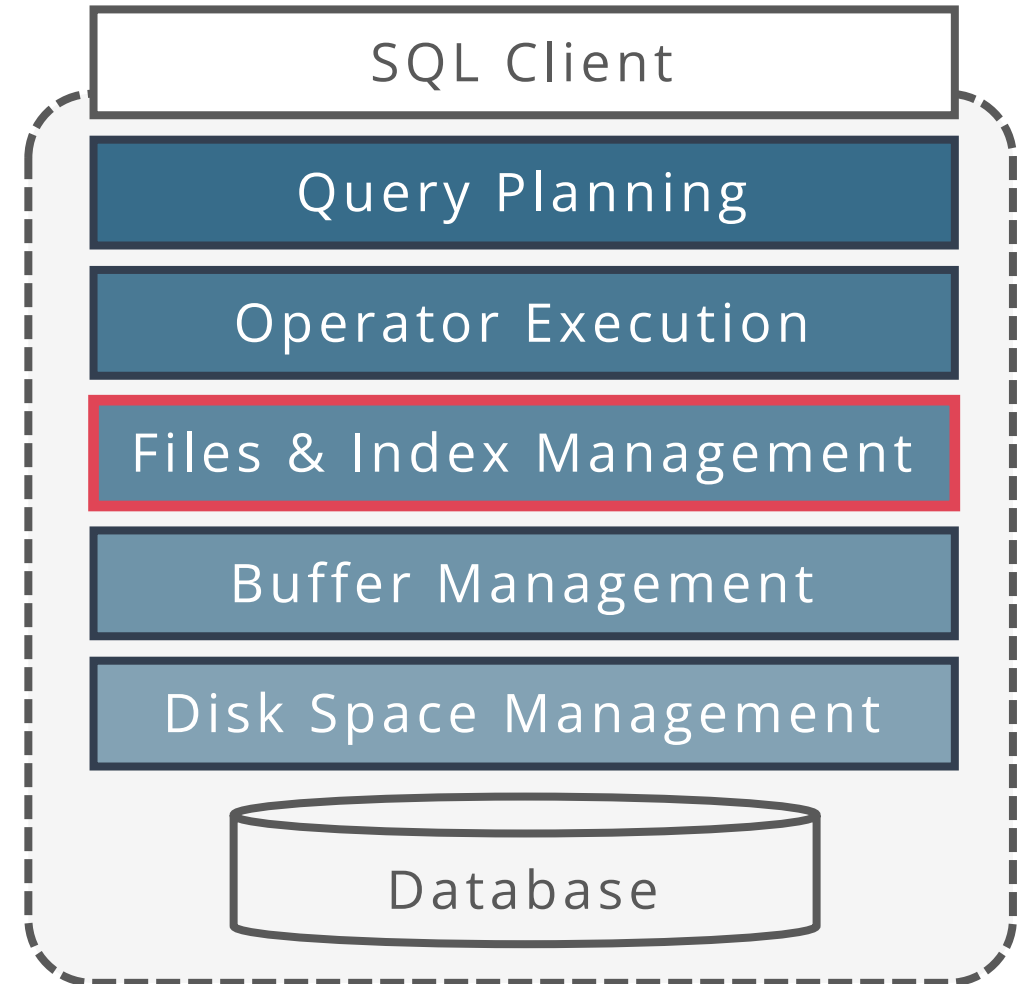

Page Directory

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.
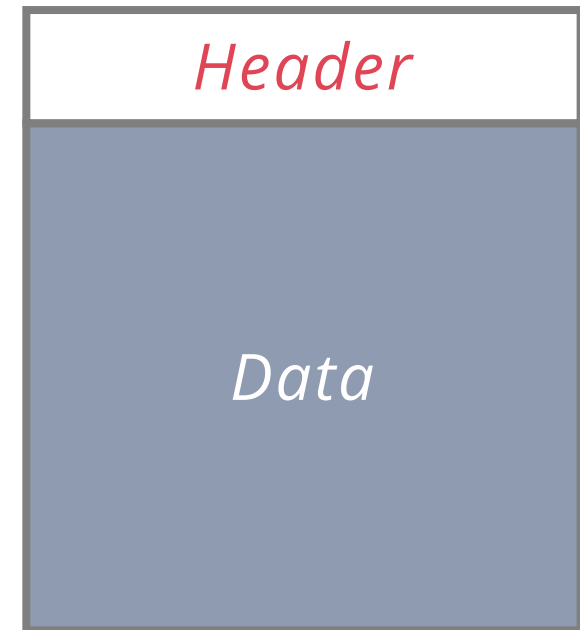
Find record by record ID, insert, delete records

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

Things to consider:

    Record length? Fixed or variable

    Page packing? Packed or unpacked

Header

Data

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)

Header

records = 4

A

B

C   D

Free   Space

Record ID:
(Page 2, 3)

Unused space

Page

Note:

Data is always stored in linear order

For presentation, we "wrap around" the linear order into a rectangle
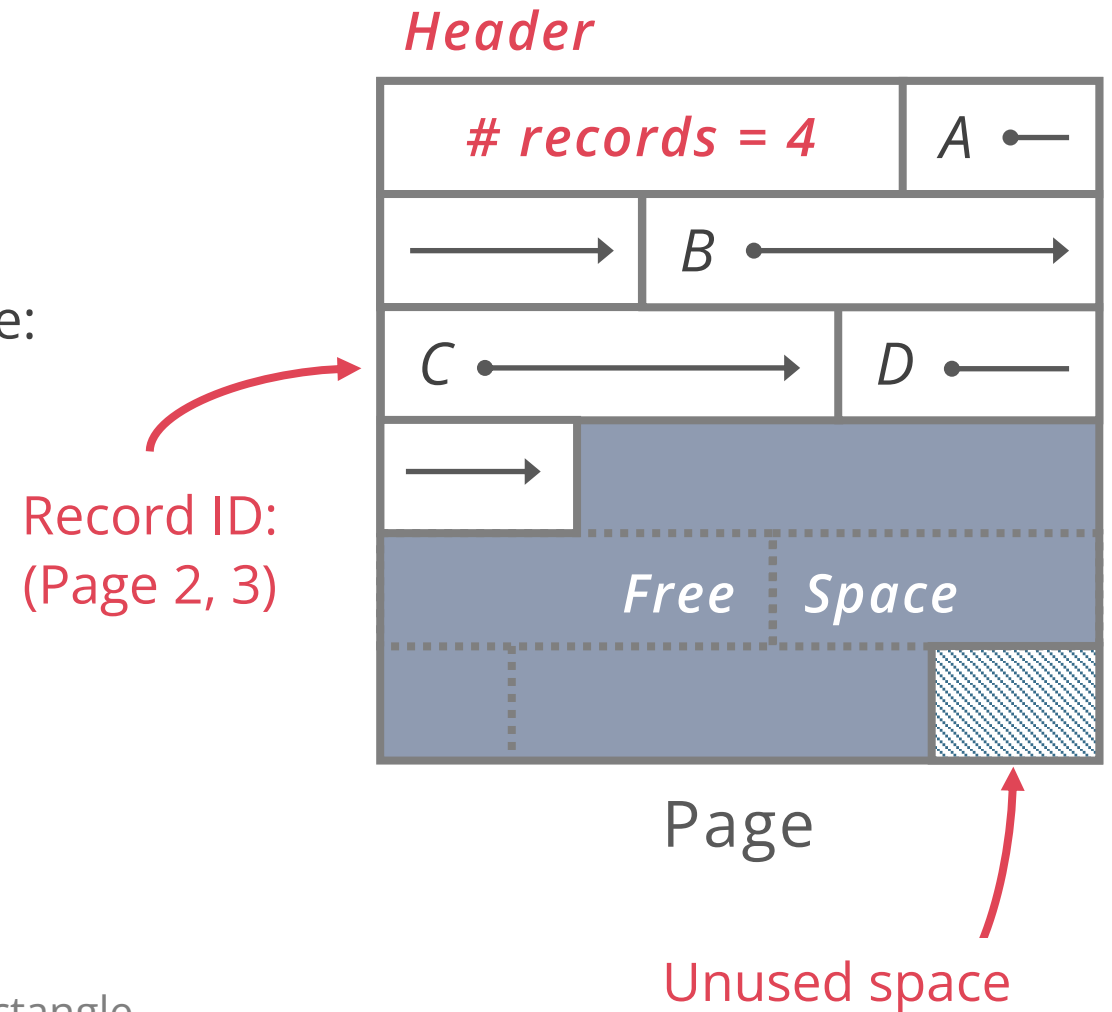
FIXED-LENGTH RECORDS, PACKED

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)

Header

records = 5

A

B

C

D

E

Free Space

Record ID:
(Page 2, 5)

Page

Unused space

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

Header

Page

Unused space

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:

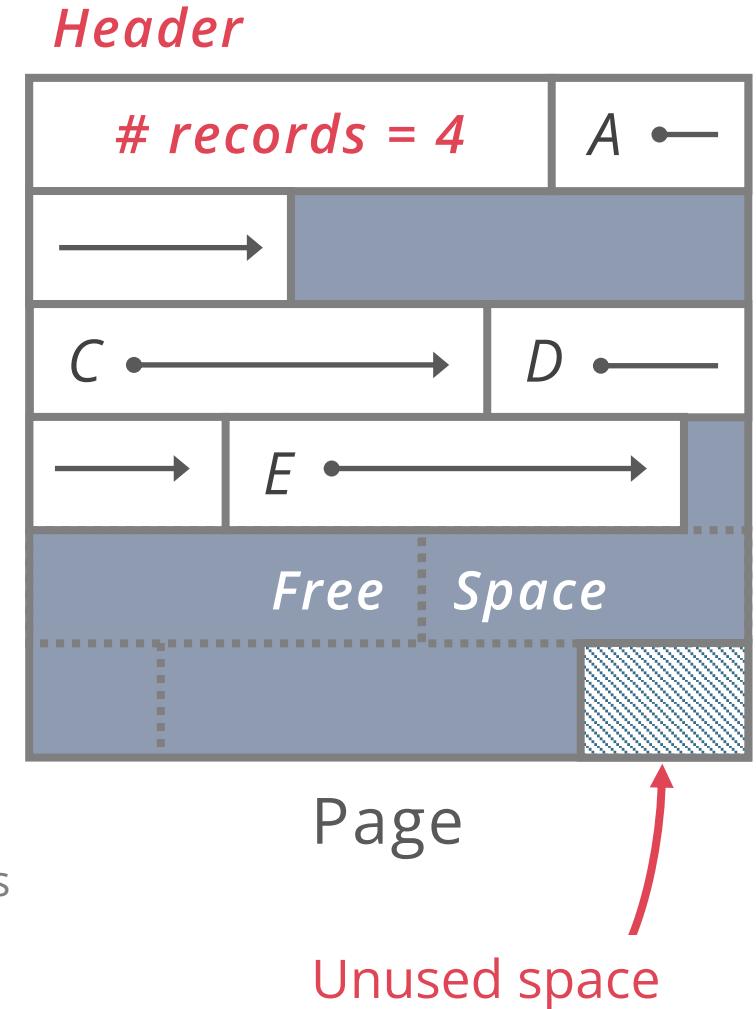*sizeof*(Header) + (position – 1) * *sizeof*(Record)

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

Header

# records = 4	*A* ←

Record ID:
(Page 2, 2)

Free   Space

Page

Unused space

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
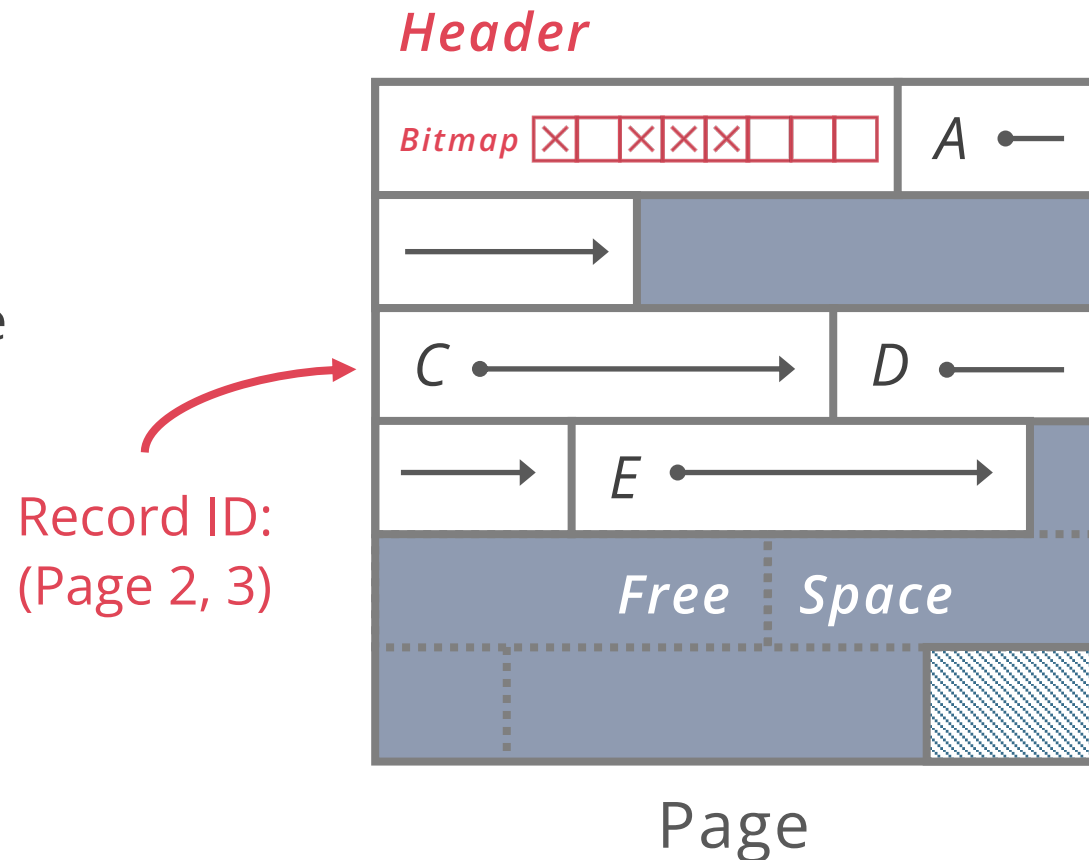
Header

Record ID:
(Page 2, 3)

Page

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

Header

Free list A

C D

E

Free Space

Page

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
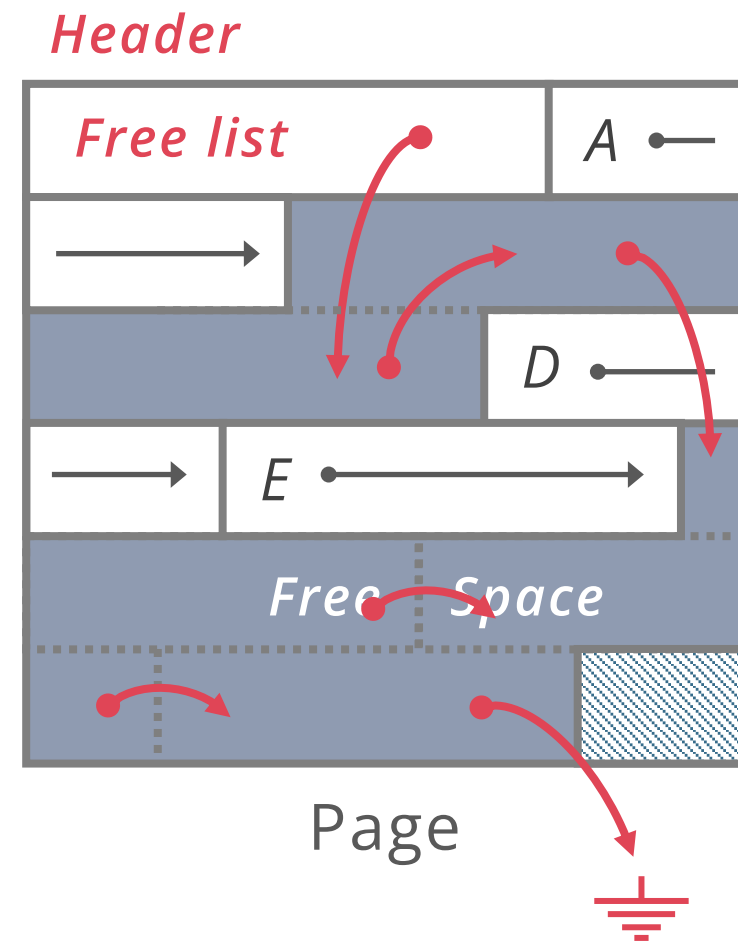
Insert record

  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

Header
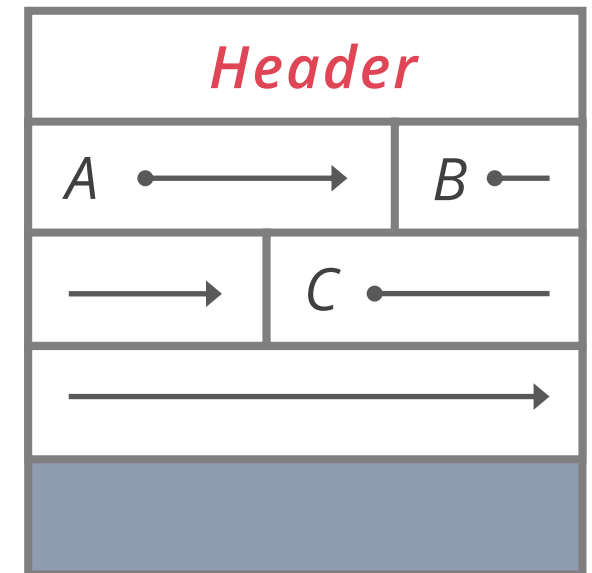
Free list

A

D

E

Free Space

Page

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**

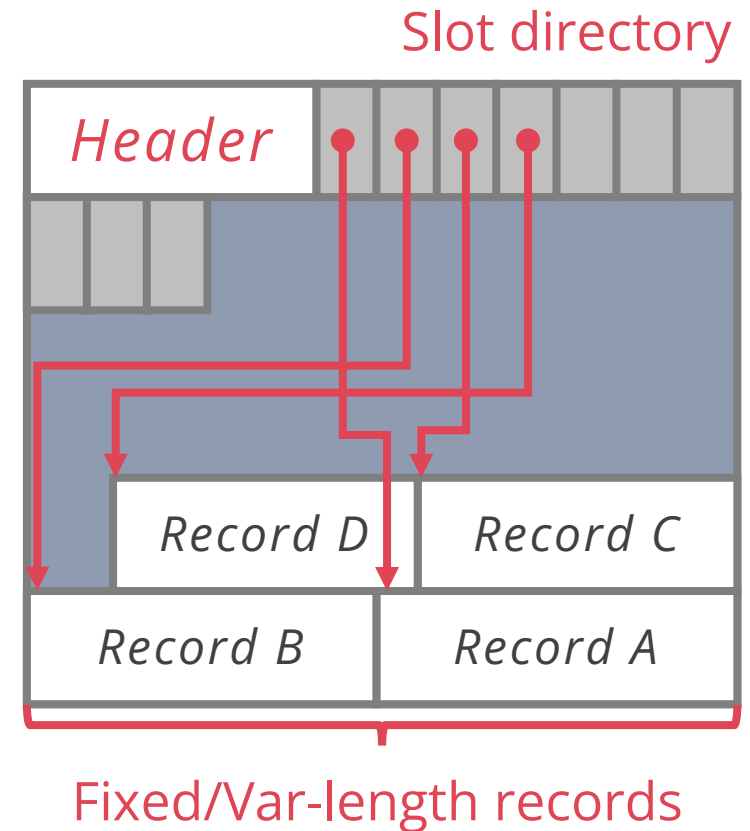Slot directory maps "slots" to the records' starting position offsets

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

Slot directory

Header

Record D   *Record C*

Record B   *Record A*

Fixed/Var-length records

SLOTTED PAGES

Records can be moved without changing rid

Delete record

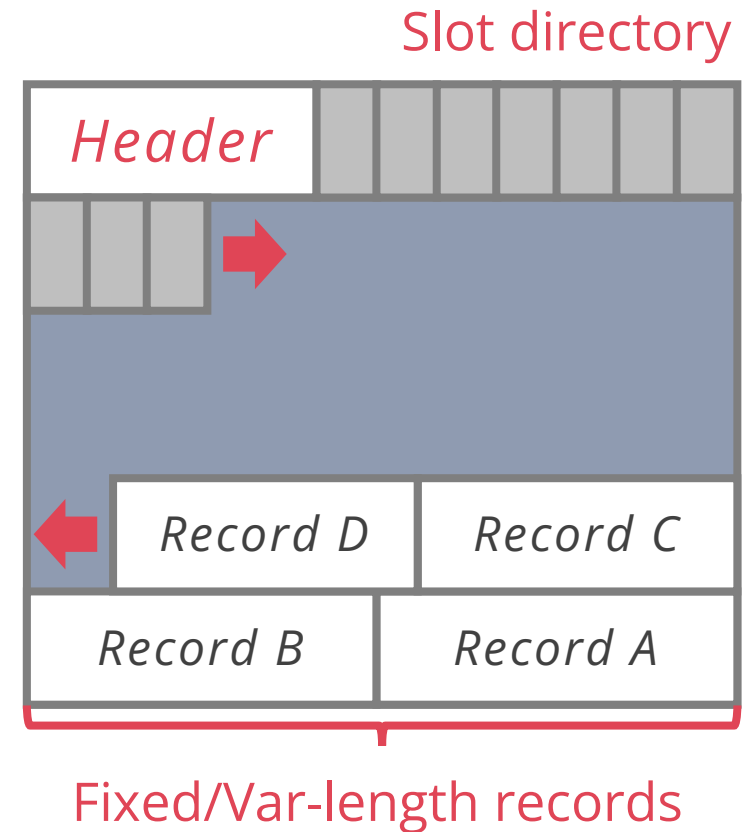Set slot offset to -1, delete slot only if last

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

Slot directory

Fixed/Var-length records

OUTLINE

Storage Media

Disk Space Management

Buffer Management

File Layout

Page Layout
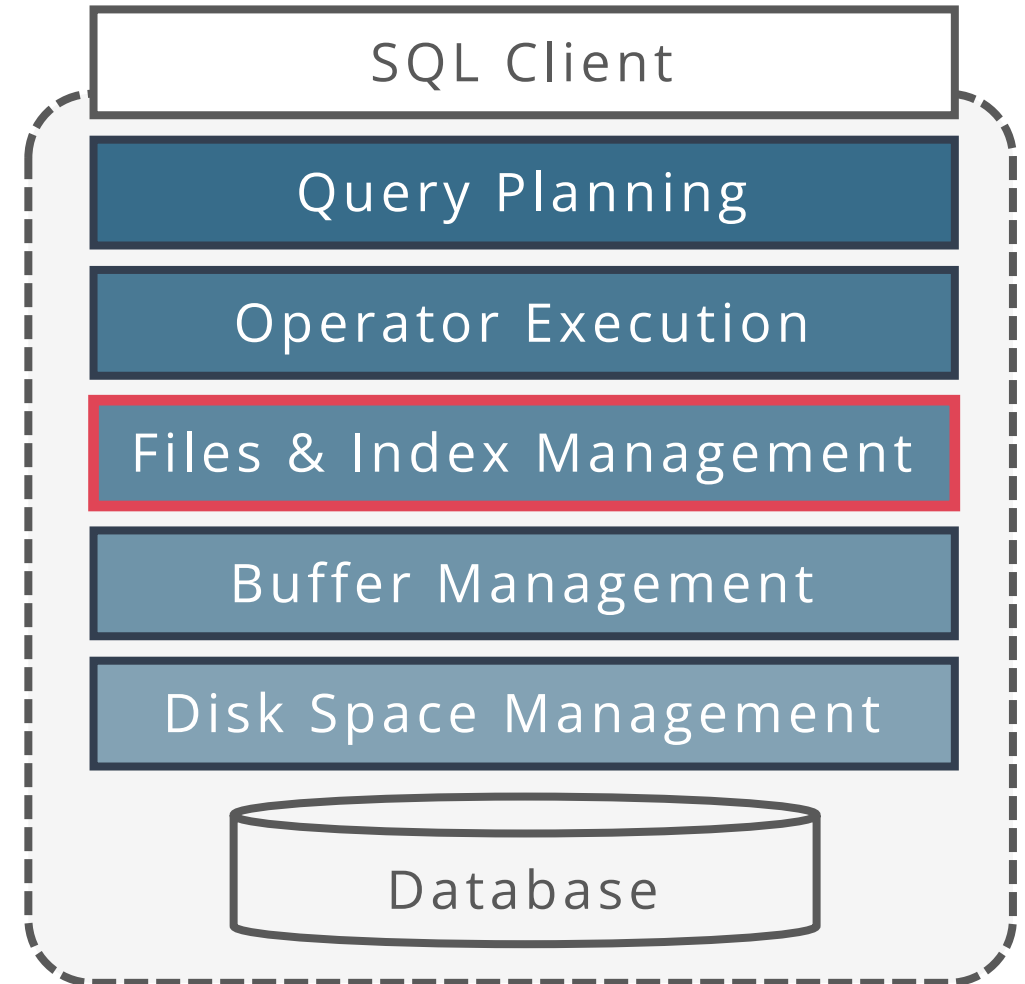
Record Layout

RECORD LAYOUT

Relational model

    Each record in table has some fixed type

We do **<u>not</u>** need to store metadata about the schema

    Information about field types is store 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

Base address

Base address + 6B

a	b	c	d
4B	2B	8B	2B

Fixed field lengths
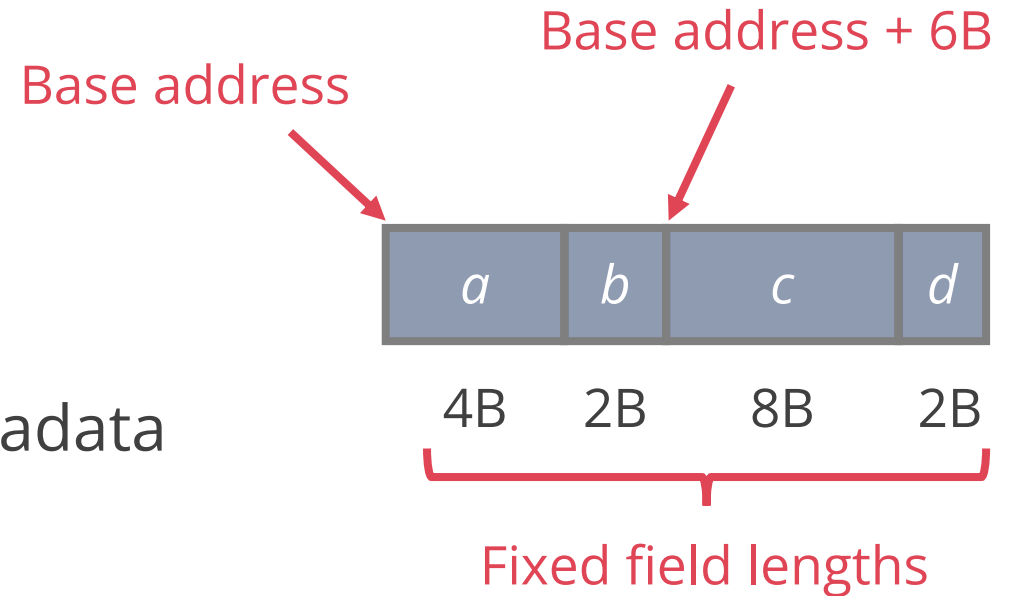
VARIABLE-LENGTH RECORDS

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"

a	$	b	$	c	$	d	$

VARIABLE-LENGTH RECORDS

2. Array of field offsets

Direct access to fields & no "escaping"

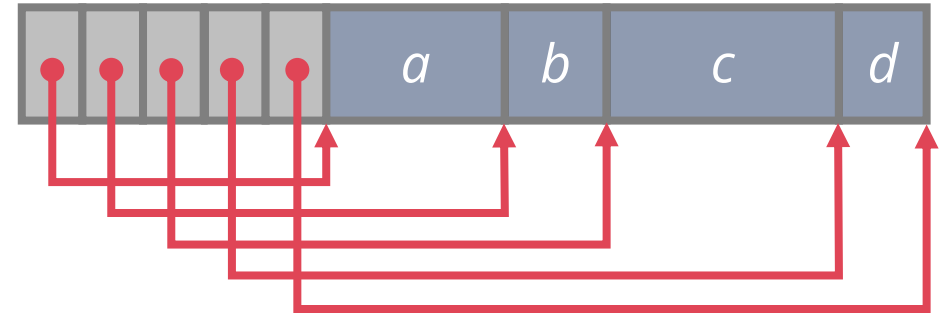Useful for fixed-length records too

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

Record

12344	Jones	CS	18
BigInt	Varchar	Varchar	Int

NULL Bitmap (stored in 1 byte)

0000

Byte Representation of Record

12344	21, 5	26, 2	18		Jones	CS

Bytes 0   8   12   16   20 21   26   28
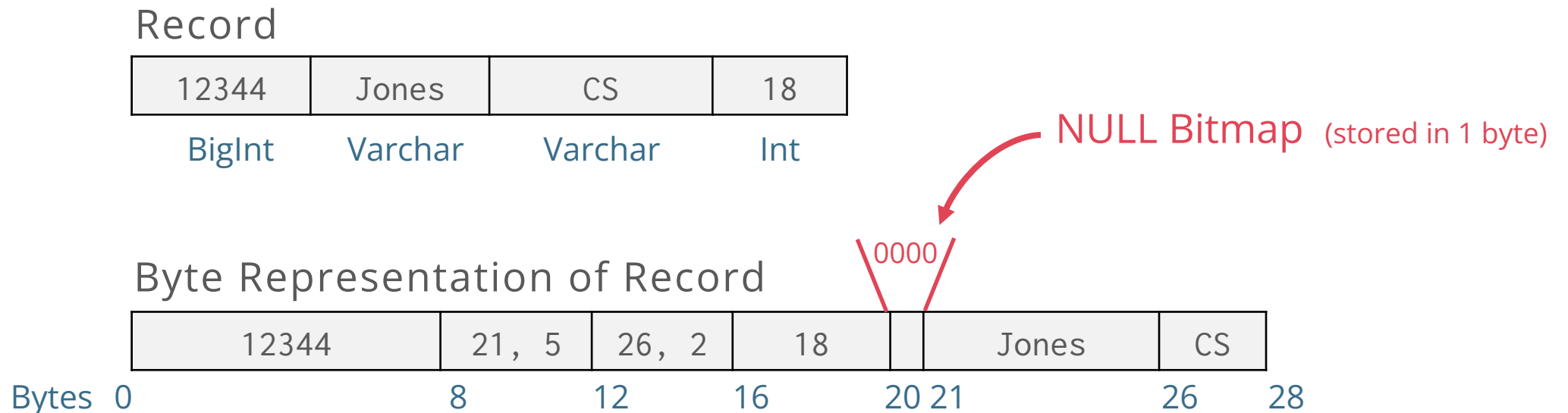
SUMMARY

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

Record

12344	Jones	CS	18
Int	Varchar	Varchar	Int

Byte Representation of Record

					a	b	c	d

Slotted Page

Header

Record #4 | Record #3
Record #2 | Record #1

Table

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

Database File

Page1	Page2
Page3	Page4
Page5	Page6