THE UNIVERSITY of EDINBURGH

Advanced Databases

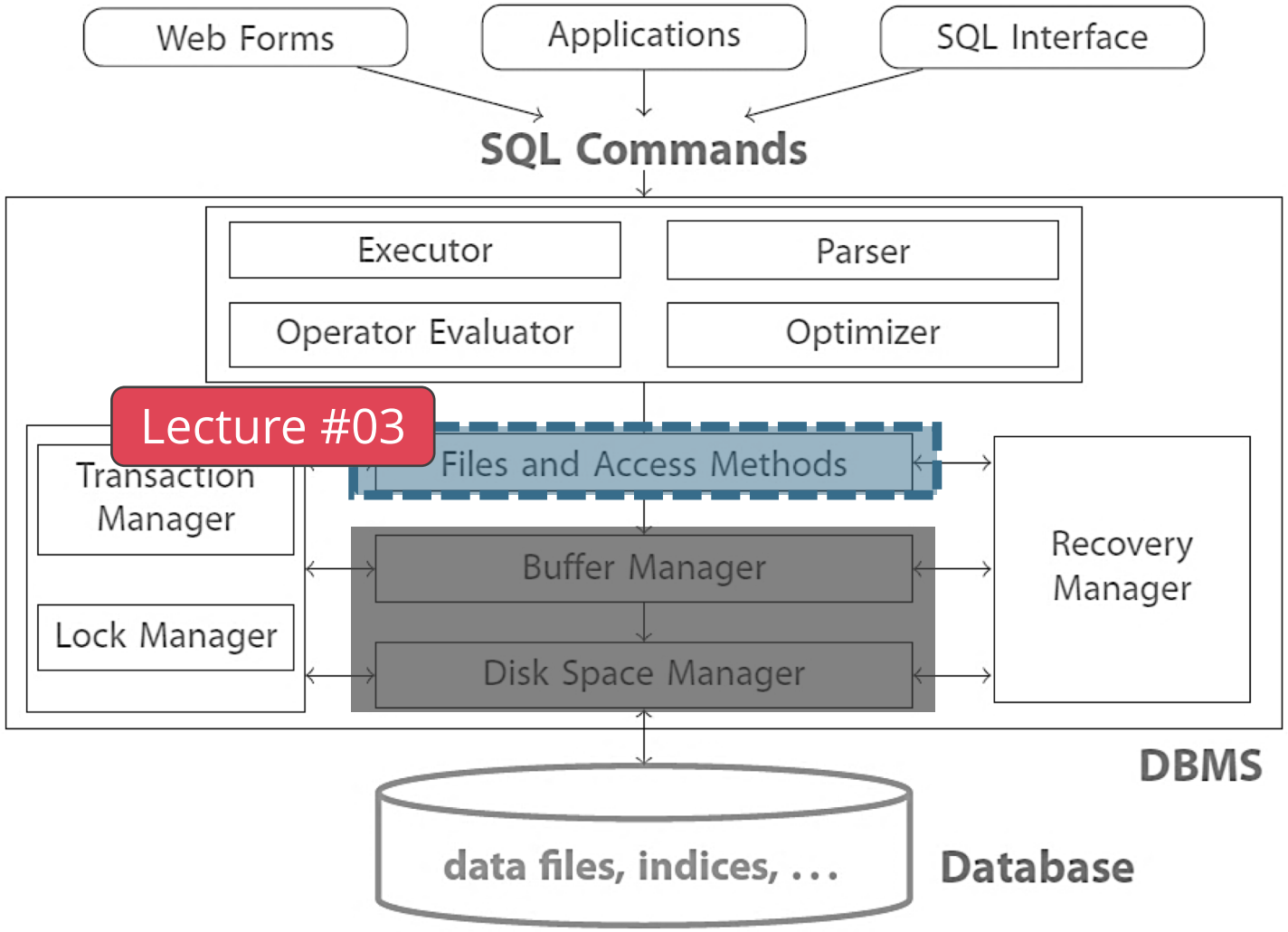
Spring 2020

Lecture #03:

Database Storage II

Milos Nikolic

DATABASE ARCHITECTURE



Agenda for Today

File layout

Page layout

Record layout

Storage models

Database Files

We have talked about pages so far. Page management is oblivious to their actual content

On the conceptual level, a DBMS primarily manages tables of records and indexes

Such tables are implemented as **files** of records

- A file consists of one or more **pages**
- Each page contains one or more **records**

Files and Access Methods

Organises data carefully to support fast access to desired subsets of records

Each record has a unique record ID

Higher-level code can:

- create/delete a file
- insert/delete/modify a record
- read a particular record
- initiate a sequential scan of all records
  - possibly with some conditions on the records to be retrieved

Types of File Organization

Different DBMSs manage pages in files on disk in different ways

- Heap File Organization
- Sequential / Sorted File Organization
- Hashing File Organization

No single file organisation responds equally fast to different ops

- Scan, lookup query, range query, insertion, deletion

At this point in the hierarchy we do not care what is page format

Database Heap

A **heap file** is an unordered collection of pages where records are stored in **no particular order**

Need metadata to keep track of what pages exist and which ones have free space

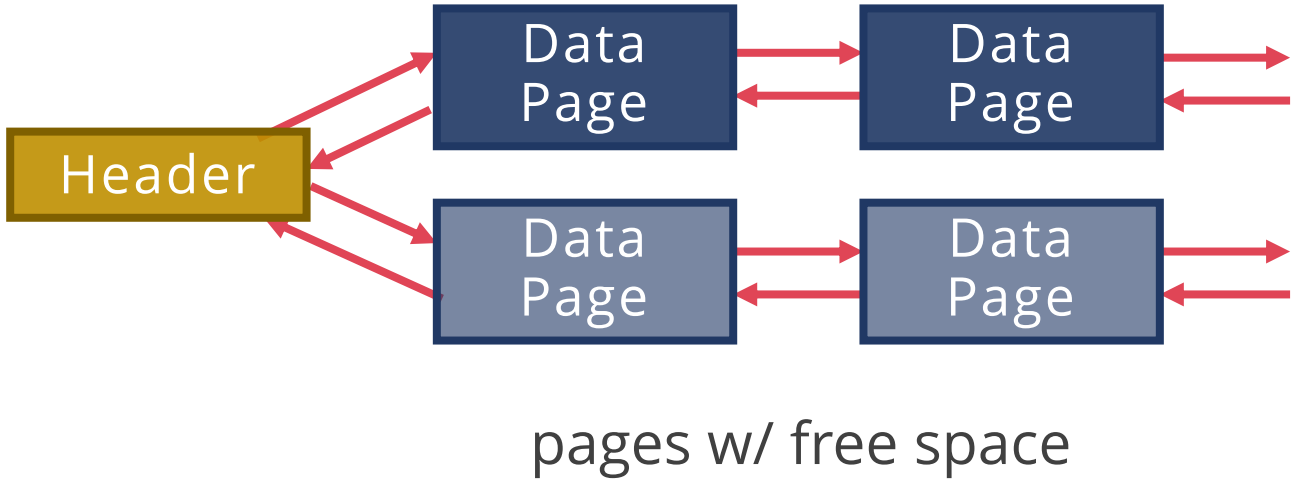
Two ways to represent a heap file:

- Linked list
- Page directory

Heap File: Linked List

Doubly linked list of pages

- Header page allocated when the file is created
- 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
- Need to search many pages to place a (large) record

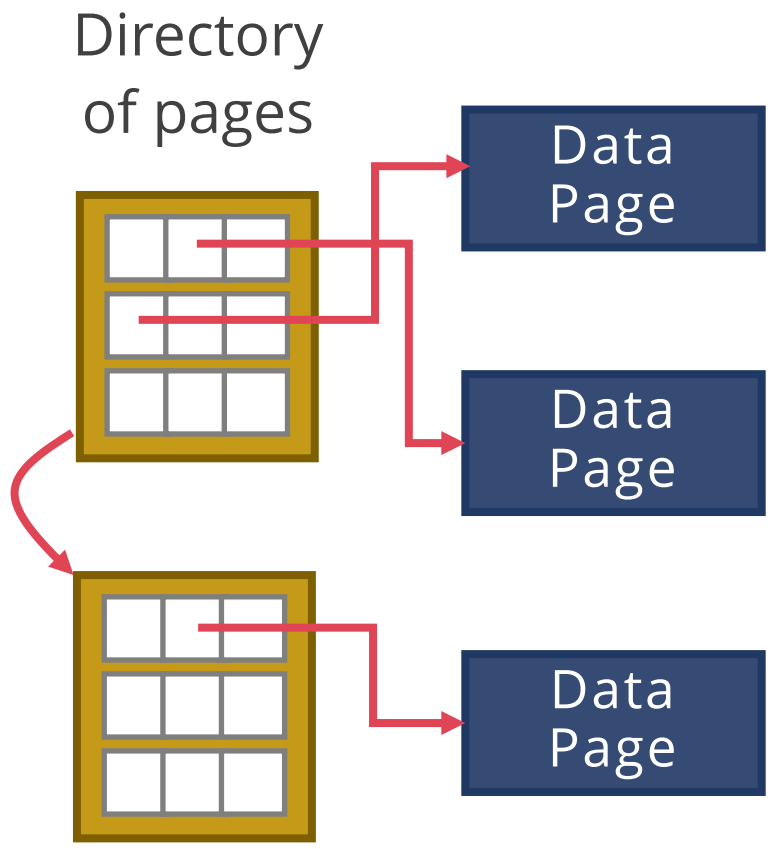
Heap File: Page Directory

DBMS maintains special pages that tracks the location of data pages in the database file

Each directory entry identifies a page

The directory records free space per page

- Free space search more efficient
- Granularity as trade-off space vs. accuracy (from bits to exact counts per entry)
- But memory overhead to host the directory



Page Format

How to organize the data stored inside the page

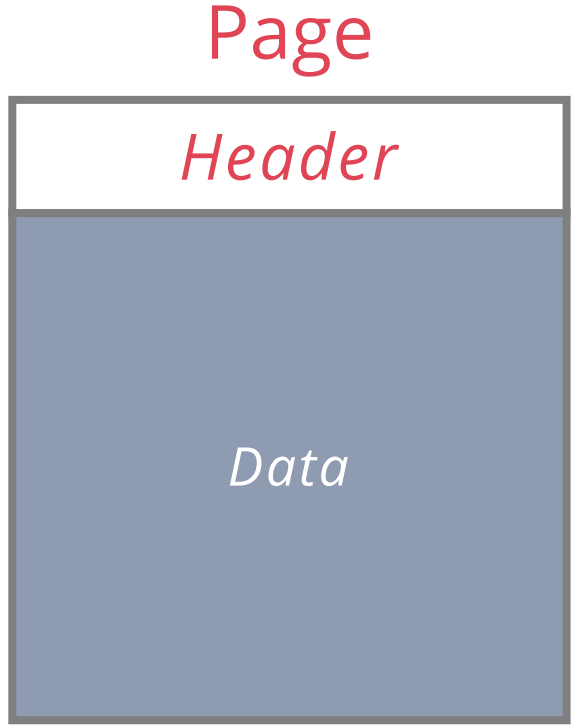
A page can be thought of as a **collection of slots**

Record ID (rid) = (pageID, slot number)

- Apps **cannot** rely on record ids to mean anything

Every page contains a header of metadata about the page's contents

- e.g., page size, checksum, DBMS version, transaction visibility, compression info



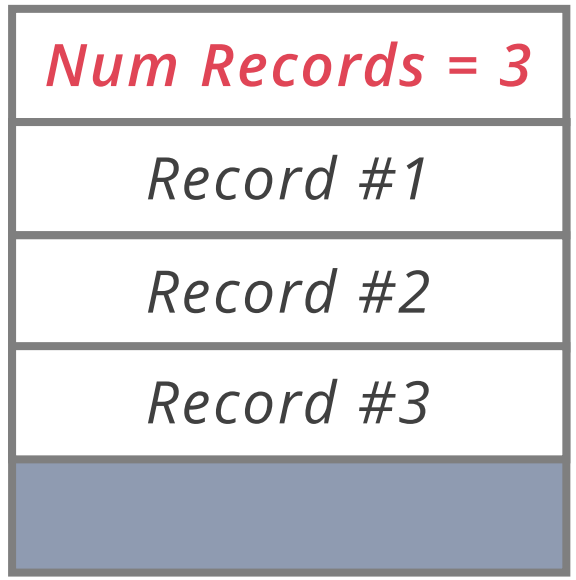
Inside a Page

How to store records in a page?

Operations: search, insert, delete records

Strawman Idea:
keep track of the number of records in a page and just append a new record to the end

- What happens if we delete a record?
  - Move the last record to the emptied slot
  - BUT this changes the last record's ID
- What happens if we have a variable-length attribute?



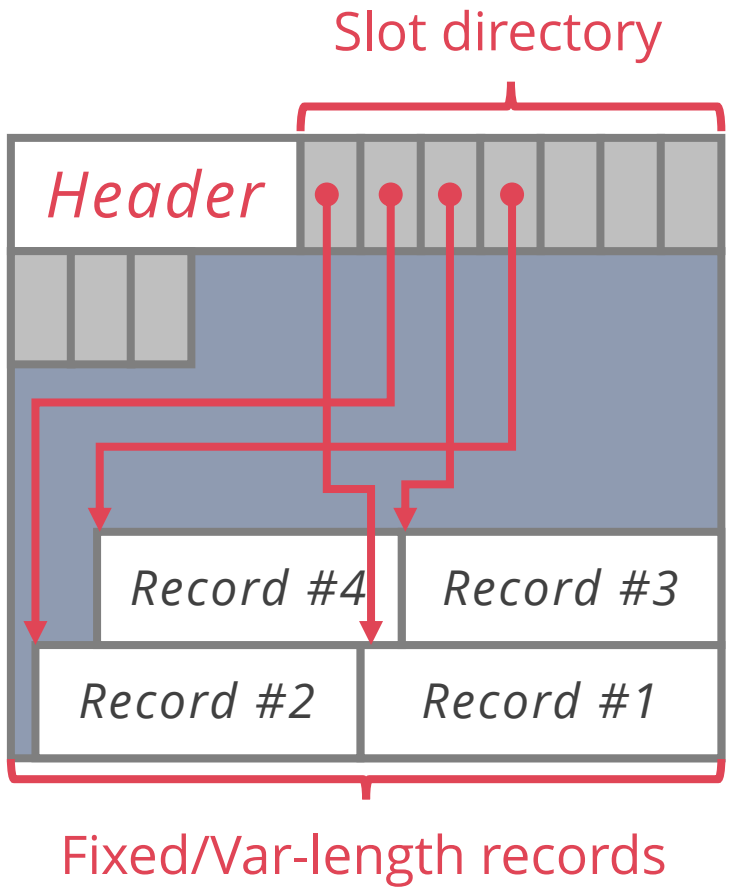
Slotted Pages

The most common layout scheme is called **slotted pages**

The slot directory maps "slots" to the records' starting position offsets

The header keeps track of

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



Slotted Pages

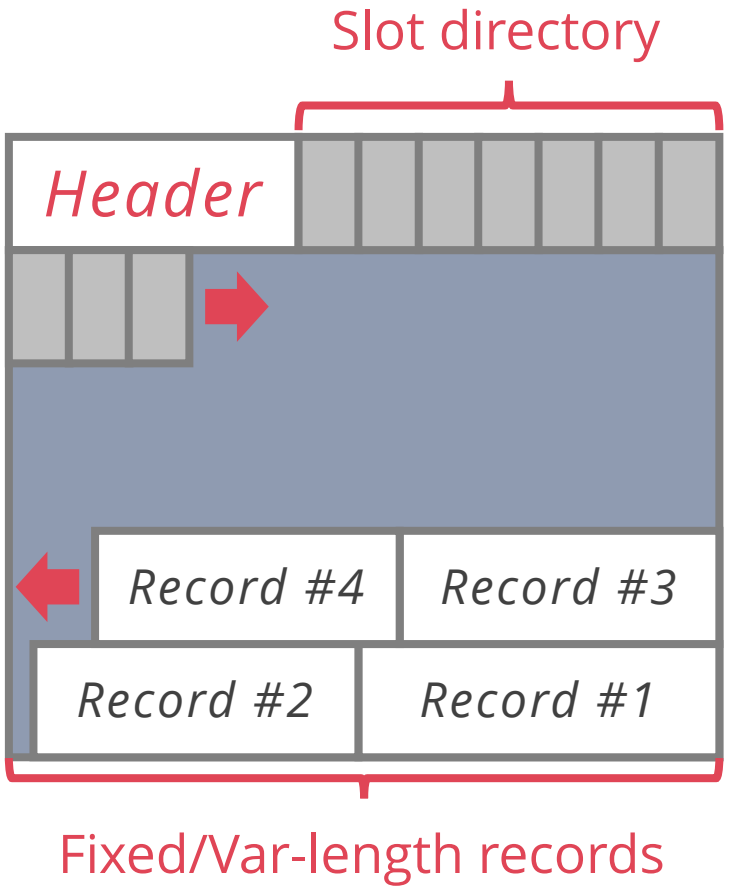
Records can be moved without changing rid

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



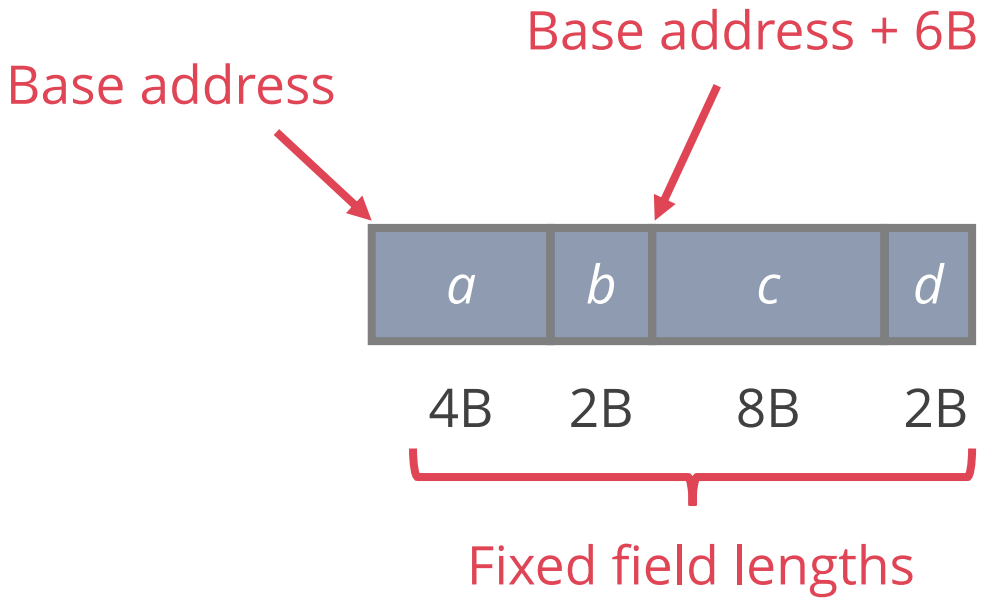
Record Format: Fixed-Length Records

Each field has a **fixed** length

Direct access to record fields

Each record can have a header storing metadata

e.g., bitmap for **NULL** values



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

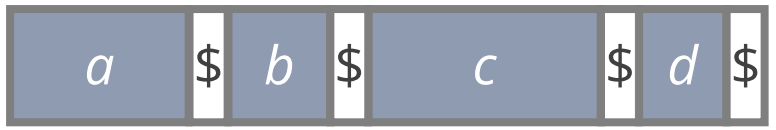
The information about field types is store in the **system catalog**

Record Format: Var-Length Records

Some fields have **variable** length

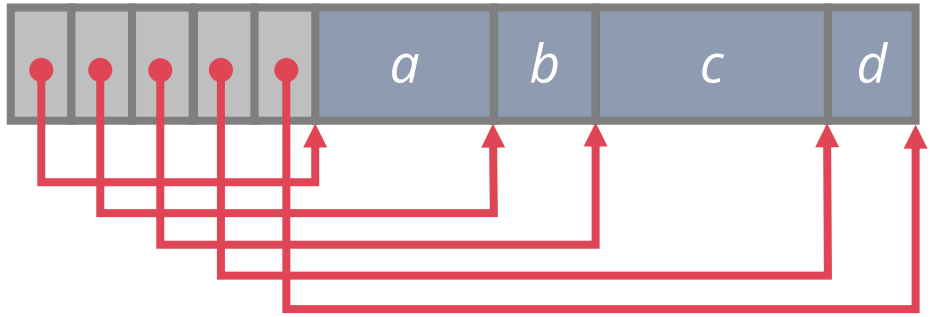
Fields delimited by special symbols

- Access to fields requires a scan of the record



Array of field offsets

- Direct access to fields
- Clean way of dealing with **NULL** values



OBSERVATION

The relational model does not specify that we have to store all of a record's attributes together in a single page

This may **not** actually be the best layout for some workloads

OLTP

On-line Transaction Processing

Simple queries that read/update a small amount of data that is related to a single entity in the database

This is usually the kind of application that people build first

```
SELECT P.*, R.*
  FROM pages AS P
 INNER JOIN revision AS R
    ON P.latest = R.revID
 WHERE P.pageID = ?
```

```
UPDATE useracct
   SET lastLogin = NOW(),
       hostname = ?
 WHERE userID = ?
```

```
INSERT INTO revisions
VALUES (?,?,?)
```

OLAP

On-line Analytical Processing

Complex queries that read large portions of the database spanning multiple entries

You execute these workloads on the data you have collected from your OLTP application(s)

```
SELECT COUNT(U.lastLogin)
       EXTRACT(month FROM
          U.lastLogin) AS month
  FROM useracct AS U
 WHERE U.hostname LIKE '%.gov'
 GROUP BY
  EXTRACT(month FROM U.lastLogin)
```

Data Storage Models

The DBMS can store records in different ways that are better for either OLTP or OLAP workloads

So far we have been assuming the **row storage model** ("n-ary storage model")

Row Storage Model

The DBMS stores values of all attributes for a single record contiguously in a page

Ideal for OLTP workloads where queries tend to operate only on an individual entity and insert-heavy workloads

Row Storage Model

The DBMS stores values of all attributes for a single record contiguously in a page

Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin

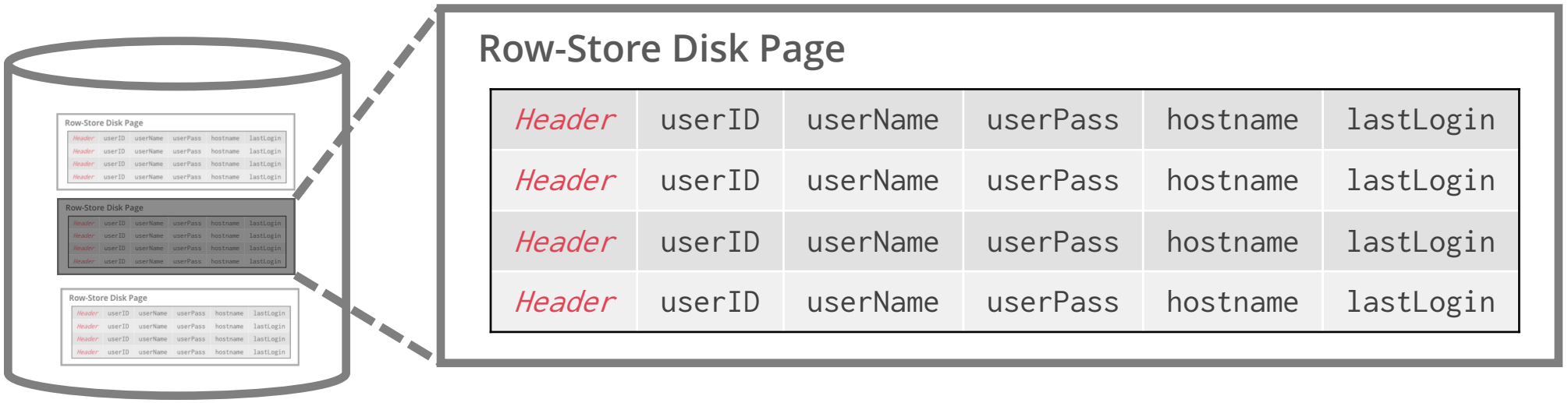
Record #1

Record #2

⋮

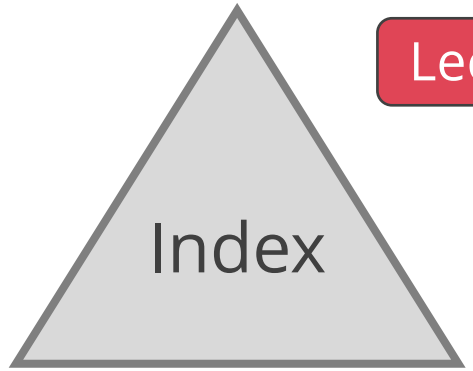
Row Storage Model

The DBMS stores values of all attributes for a single record contiguously in a page



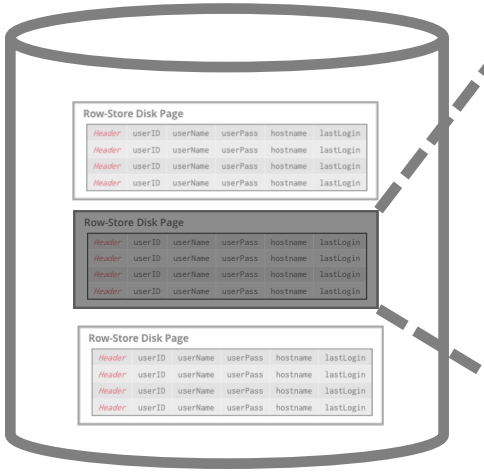
Row Storage Model

```
SELECT * FROM useracct
 WHERE userName = ?
   AND userPass = ?
```



Lecture #04

Touches small amounts of data



Row-Store Disk Page

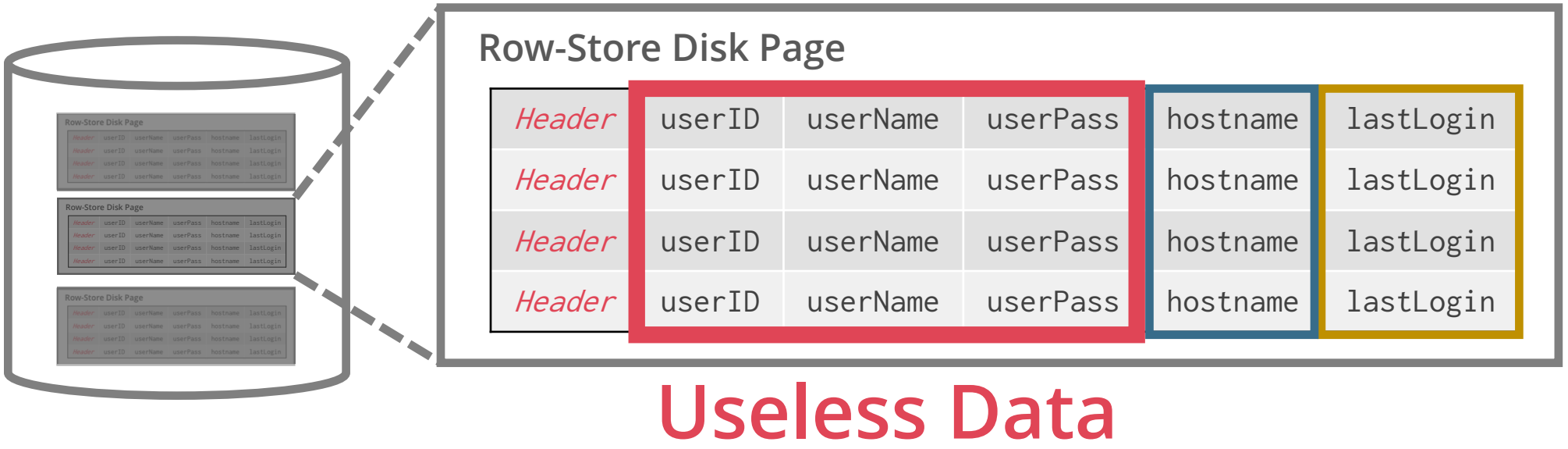
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin

Row Storage Model

```
SELECT COUNT(U.lastLogin),
       EXTRACT(month FROM U.lastLogin) AS month
  FROM useracct AS U
 WHERE U.hostname LIKE '%.gov'
 GROUP BY EXTRACT(month FROM U.lastLogin)
```

Scans entire relation

Most read data not needed



Row Storage Model

Advantages

Fast inserts, updates, and deletes

Good for queries that need the entire record

Disadvantages

Not good for scanning large portions of the table and/or a subset of the attributes

COLUMNAR STORAGE MODEL

The DBMS stores the values of a single attribute for all records contiguously in a page

- Also known as “decomposition storage model”

Ideal for OLAP workloads where read-only queries perform large scans over a subset of the table’s attributes

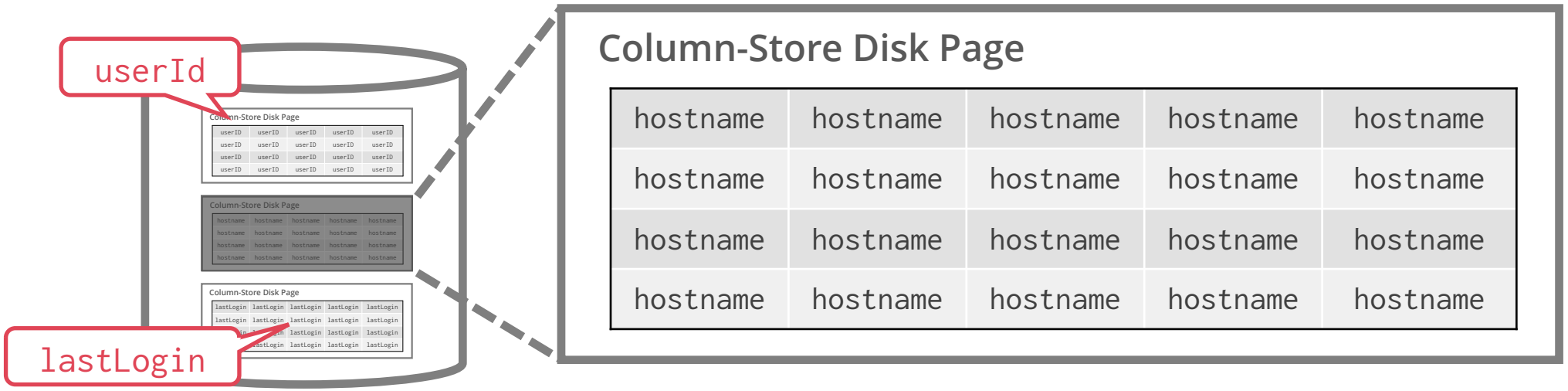
Columnar Storage Model

The DBMS stores the values of a single attribute for all records contiguously in a page

Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin
Header	userID	userName	userPass	hostname	lastLogin

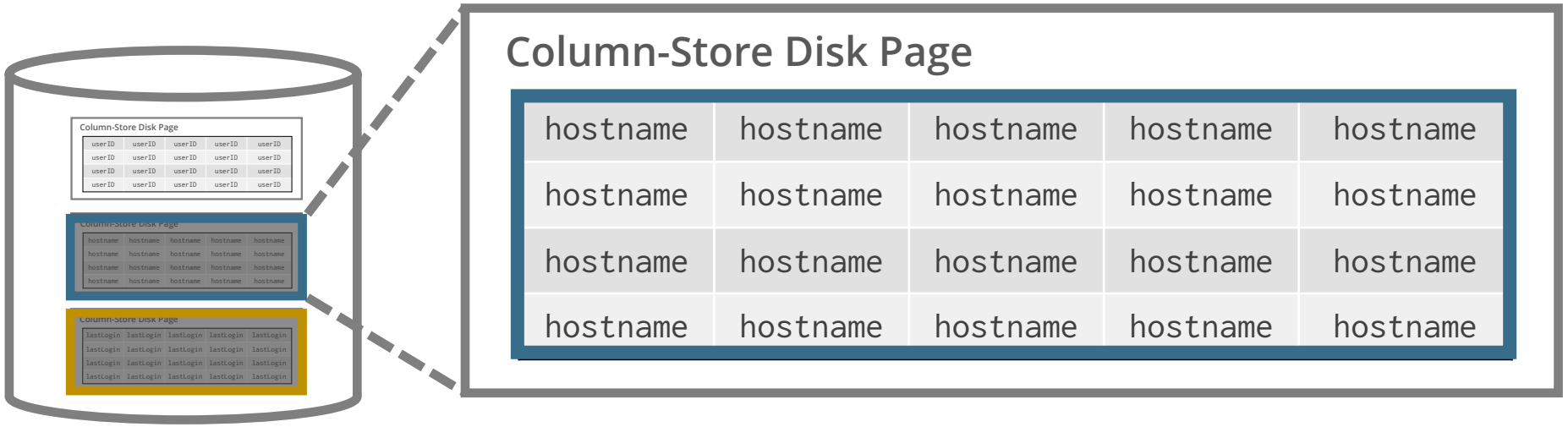
Columnar Storage Model

The DBMS stores the values of a single attribute for all records contiguously in a page



Columnar Storage Model

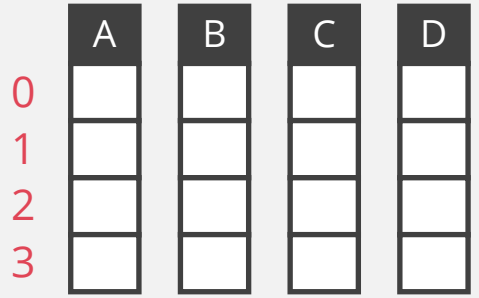
```
SELECT COUNT(U.lastLogin),
       EXTRACT(month FROM U.lastLogin) AS month
  FROM useracct AS U
 WHERE U.hostname LIKE '%.gov'
 GROUP BY EXTRACT(month FROM U.lastLogin)
```



Record Identification

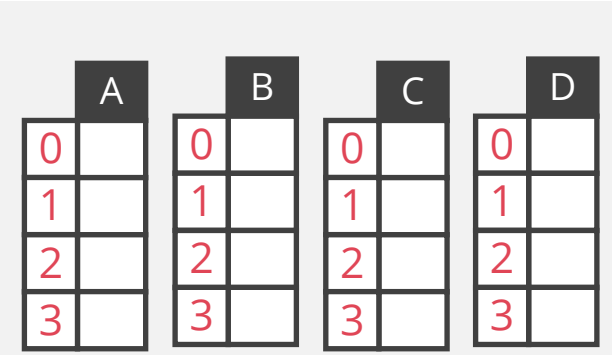
Choice #1: Fixed-length Offsets

Each value is the same length for an attribute



Choice #2: Embedded Record Ids

Each value is stored with its record id in a column



Columnar Storage Model

Advantages

Reduces the amount of wasted I/O because the DBMS only reads the data that it needs

Better query processing and data compression

Disadvantages

Slow for point queries, inserts, updates, and deletes because of record splitting / stitching

Most DBMSs now support the columnar storage model

Conclusion

Database is organized in pages

Different ways to track pages

Different ways to store pages

Different ways to store records

Important to choose the right storage model for the target workload

- OLTP = Row store
- OLAP = Column store