CSE 444: Database Internals

Section 2: Indexing

Plan for the Sections

- We will go through examples together
- Should be a good practice for the homework problems
- Ideas, suggestions, comments, feedback are always welcome
  - write your thoughts on discussion board

Scenario

Consider the following database schema:

Field Name	Data Type	Size on disk
Id (primary key)	Unsigned INT	4 bytes
firstName	Char(50)	50 bytes
lastName	Char(50)	50 bytes
emailAddress	Char(100)	100 bytes

Scenario

Total records in the database = 5,000,000
Length of each record = 4+50+50+100 = 204 bytes

Let the default block size be 1,024 bytes

Therefore,
We will have 1024/204 = 5 records per disk block
Also, No. of blocks needed for the entire table = 5000000/5 = 1,000,000 blocks

Scenario

Suppose you want to find the person with a particular id (say 5000)

What is the best way to do so?

Scenario

Linear Search

No. of block accesses = 1000000/2 = **500,000**

Binary Search

No. of block accesses = $\log_2$ 1000000 = 19.93 = **20**

Scenario

Now, suppose you want to find the person having firstName = ‘John’

Here, the column isn’t sorted and does not hold an unique value.

What is the best way to do search for the records?

Scenario

Solution: Create an index on the firstName column

The schema for an index on firstName is:

Field Name	Data Type	Size on disk
firstName	Char(50)	50 bytes
(record pointer)	Special	4 bytes

Scenario

Total records in the database = 5,000,000
Length of each index record = 4+50 = 54 bytes

Let the default block size be 1,024 bytes

Therefore,
We will have 1024/54 = 18 records per disk block
Also, No. of blocks needed for the entire table = 5000000/18 = 277,778 blocks

Scenario

Now, a binary search on the index will result in $\log_2 277778 = 18.08 = 19$ block accesses.

Also, to find the address of the actual record, which requires a further block access to read, bringing the total to 19 + 1 = **20 block accesses**.

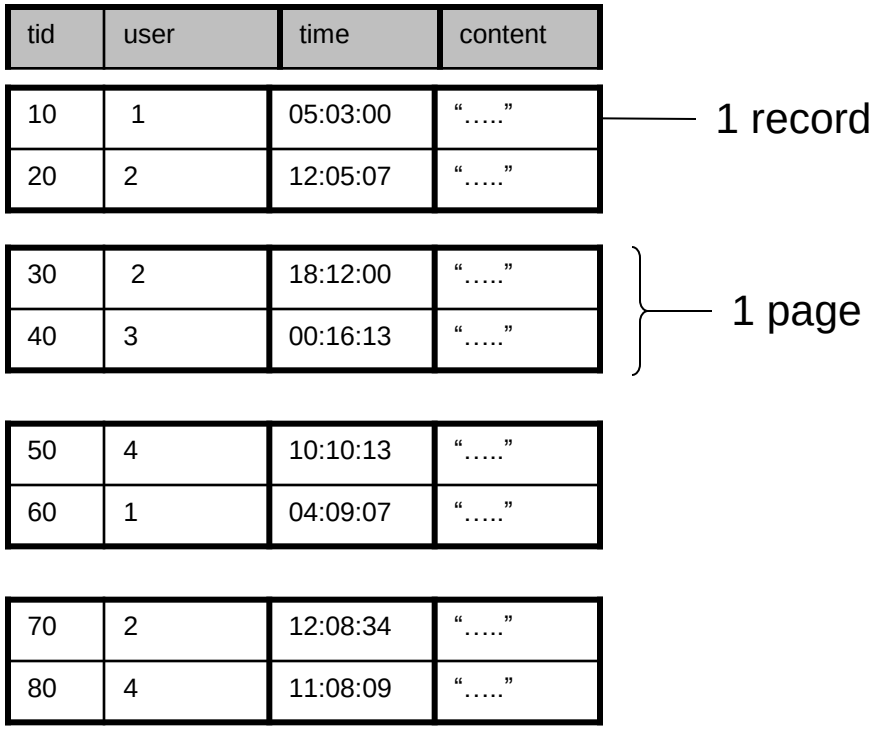
Thus, indexing results in a much better performance as compared to searching the entire database.

Indexes: Useful for search query / range query / joins

Revisit Tweet Example:

Tweets(tid, user, time, content)

Tweet Relation in a Sequential File

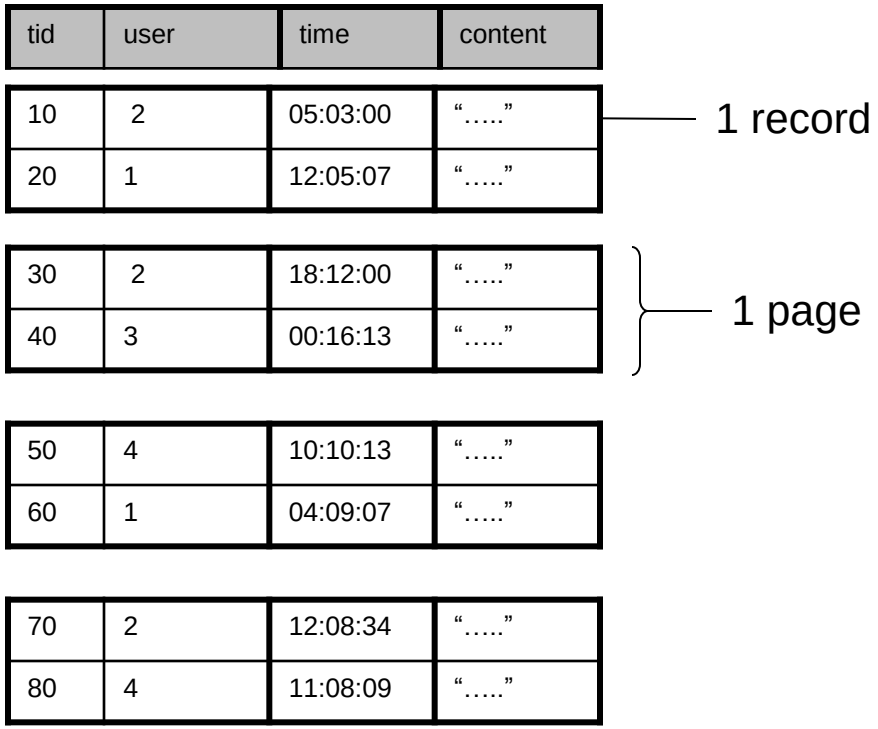


- File is sorted on “tid”

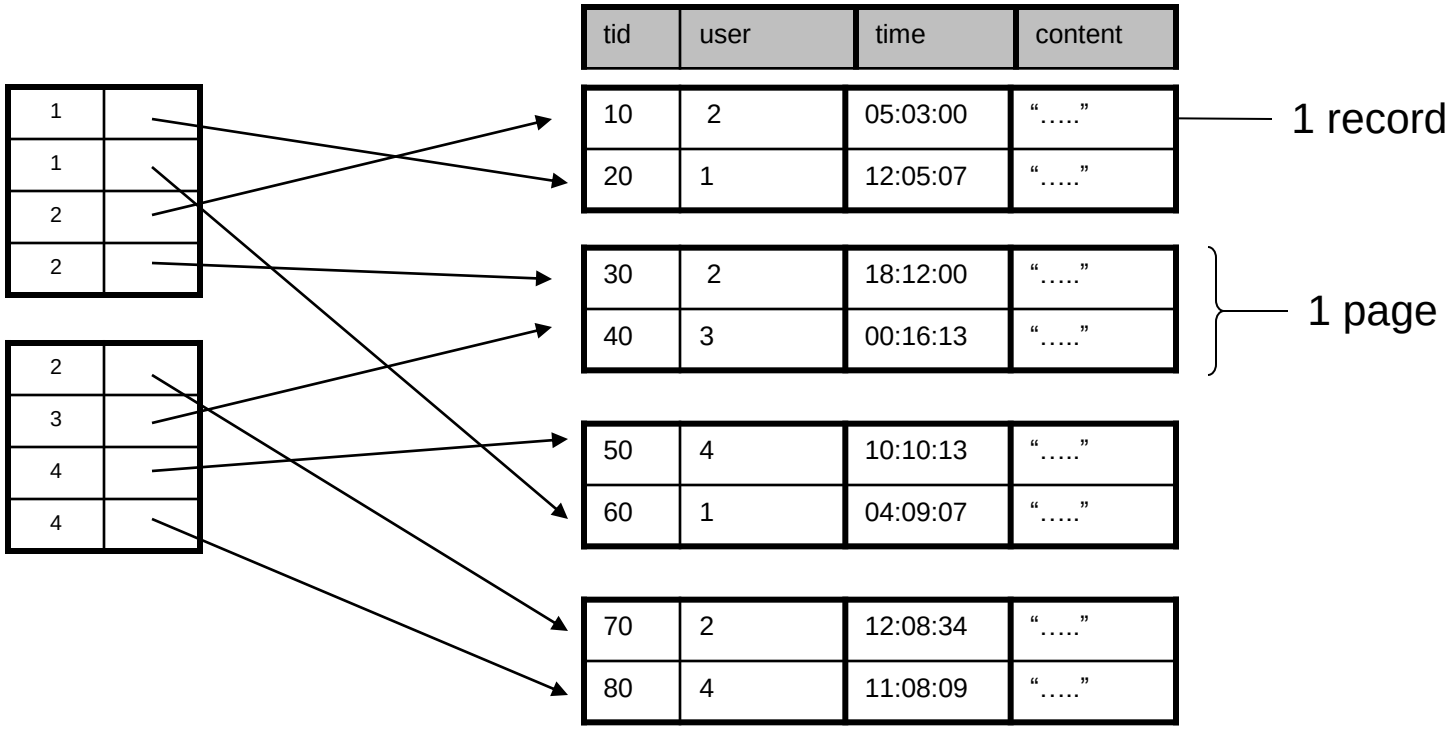
Index Classification

- Primary/Secondary
- Dense/Sparse
- Clustered/Unclustered
- Question: Draw a secondary dense index on "user"

Ex1. Secondary Dense Index

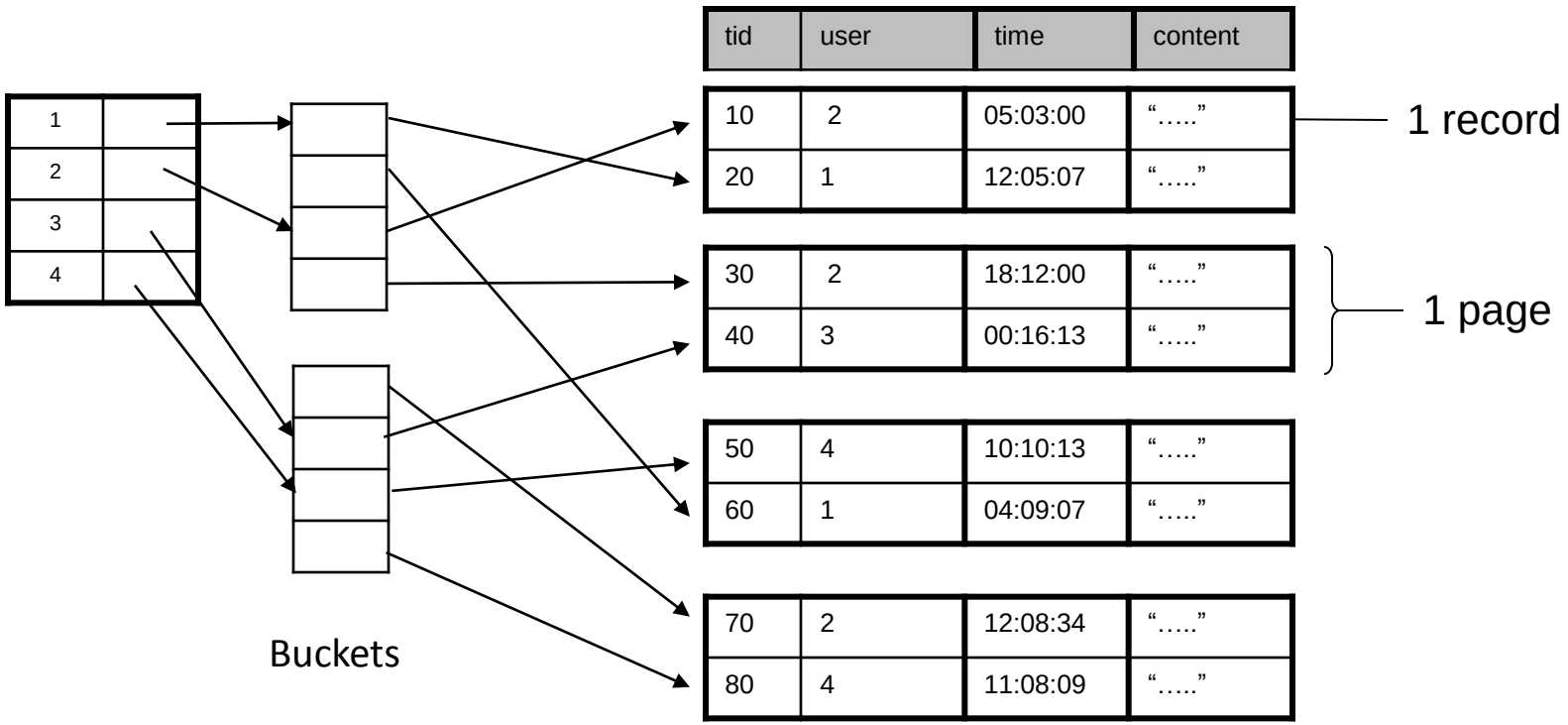


Ex1. Secondary Dense Index (user)



- Dense: an “index key” (not database key) for every database record
- Secondary: cannot reorder data, does not determine data location
- Also, Unclustered: records close in index may be far in data

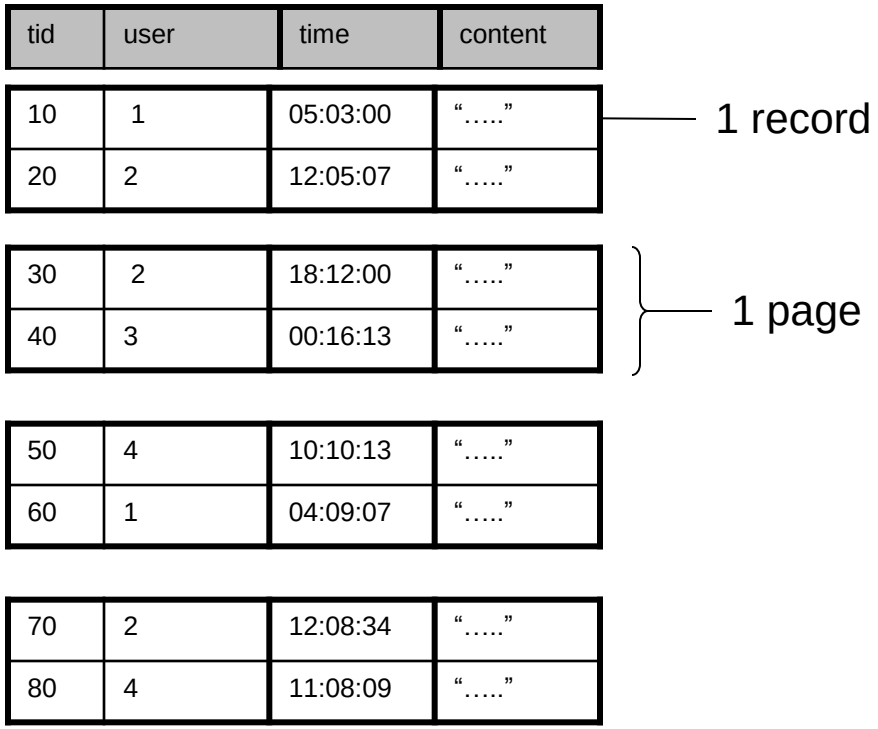
Ex1. Alternative solution



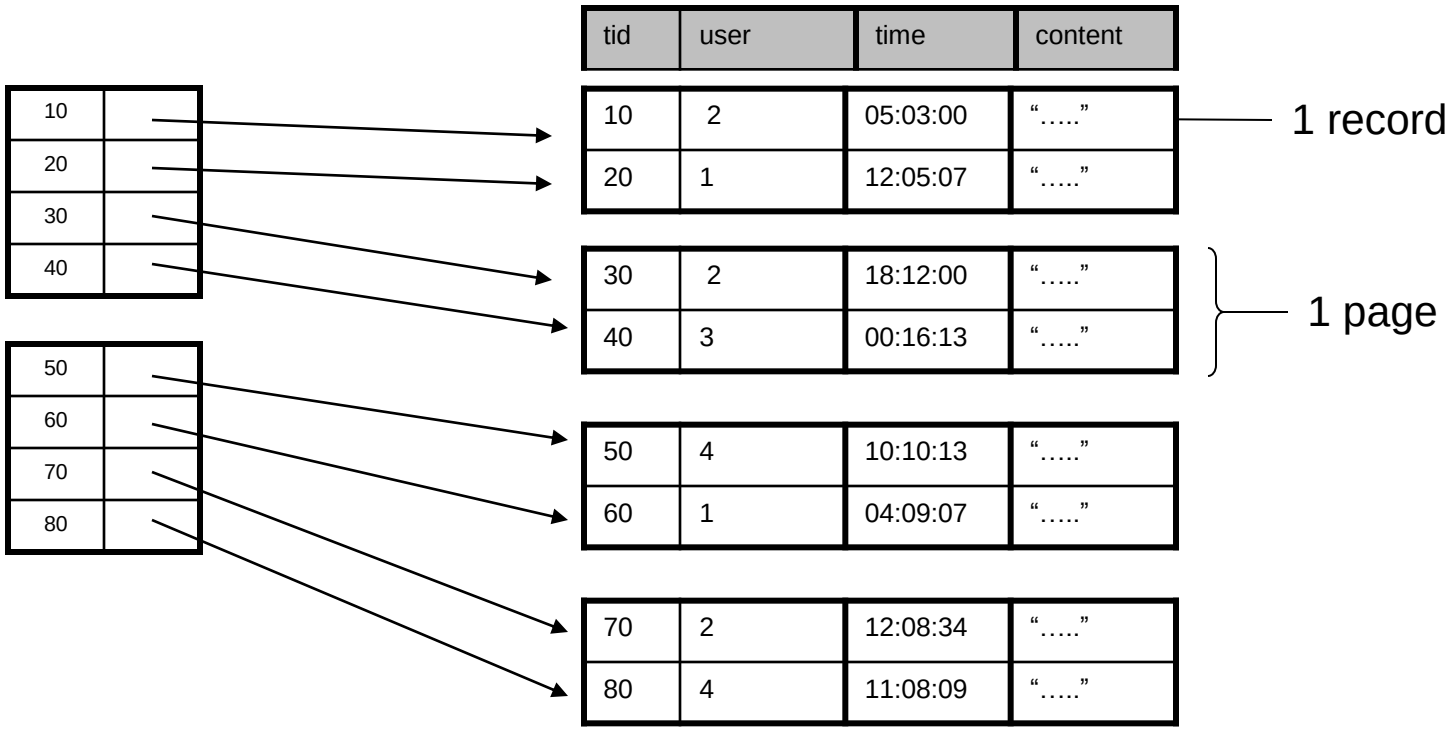
- Convenient way to avoid repeating values and saving space is to use a level of indirection, called *buckets*, between the secondary index file and the data file

- Question: Draw a primary dense index on "tid"

Ex2. Primary Dense Index (tid)



Ex2. Primary Dense Index (tid)



- Dense: an "index key" for every database record
  - (In this case) every "database key" appears as an "index key"
- Primary: determines the location of indexed records
- Also, Clustered: records close in index are close in data

Primary Clustered Index

Vs.

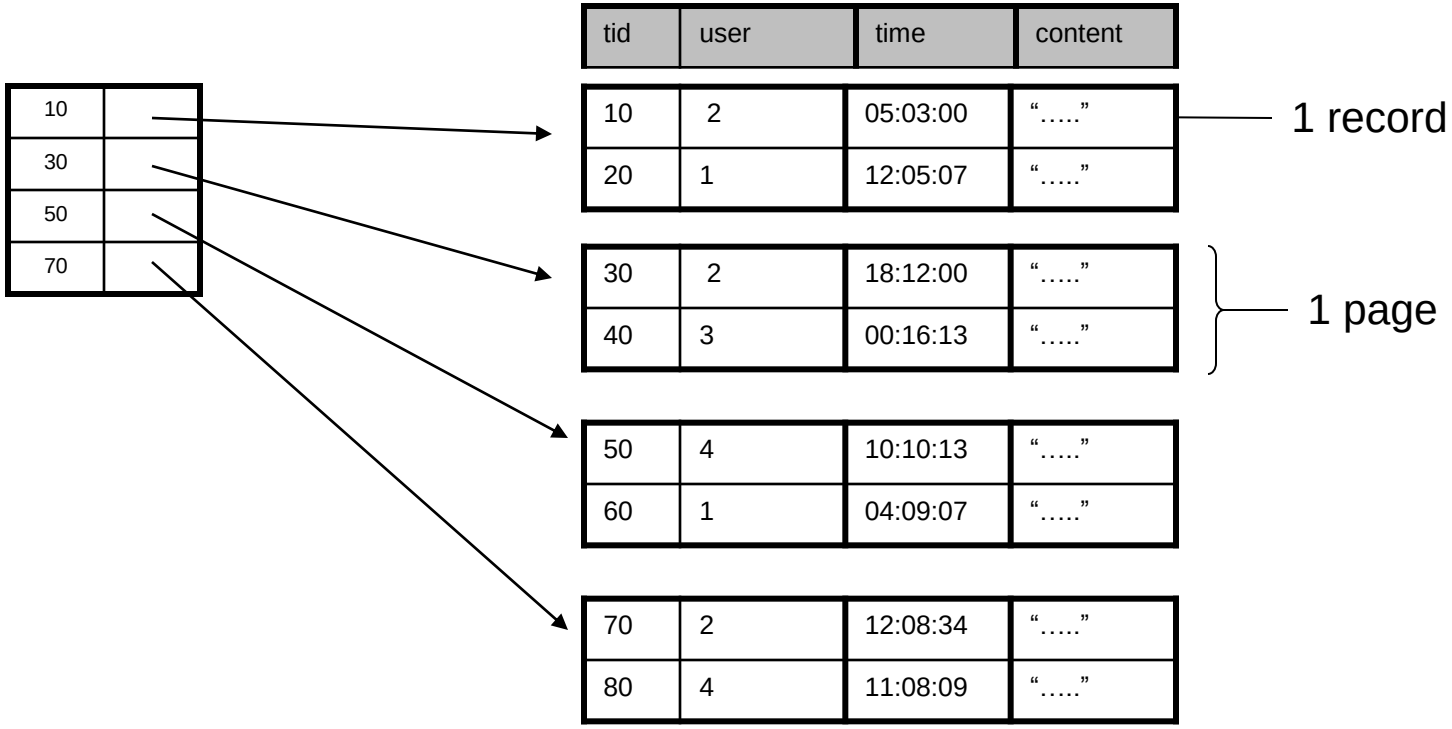
Secondary Unclustered Index?

Clustered Index can be made Sparse

(normally one key per page)

- Question: Draw a primary sparse index on "tid"

Ex3. Primary Sparse Index (tid)

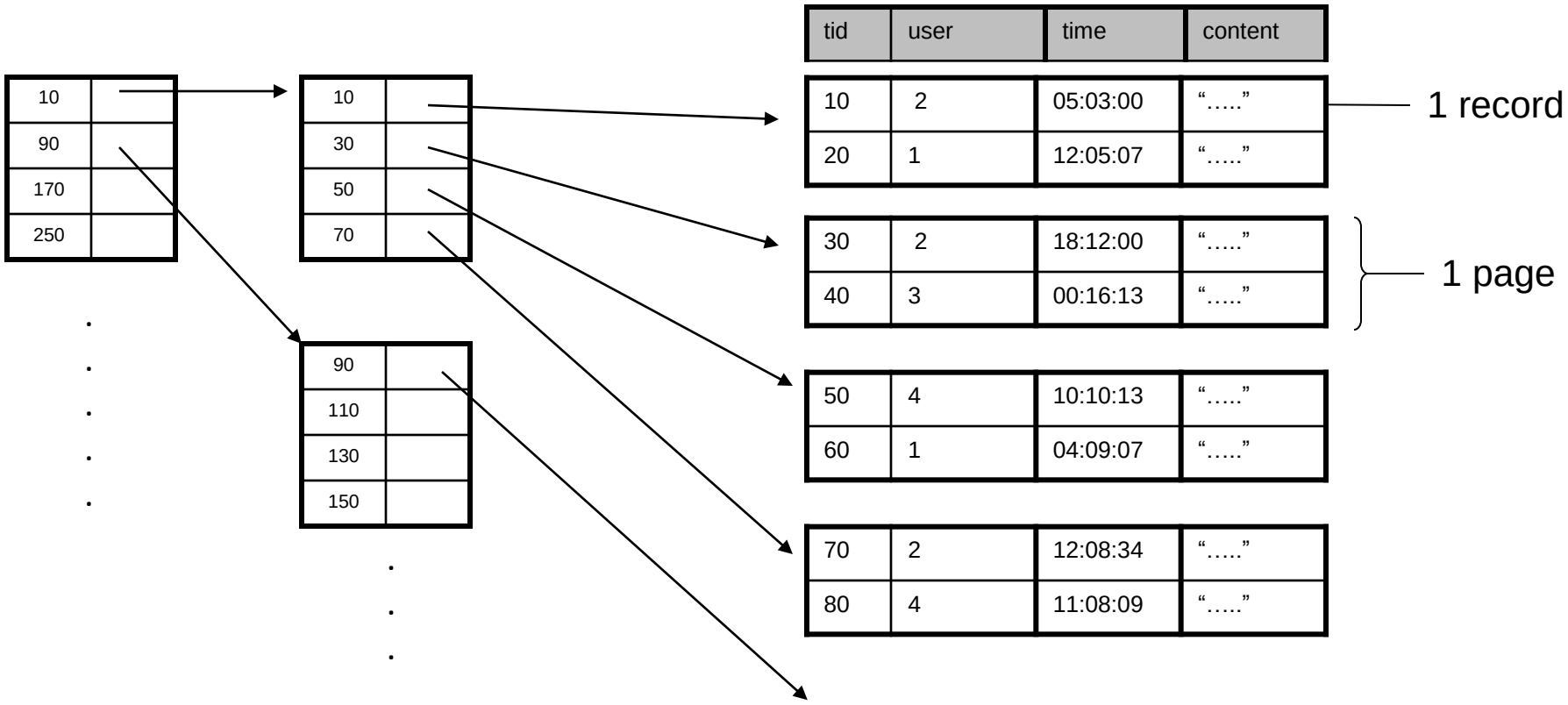


- Only one index file page instead of two

Discussion

- Primary/Secondary
  - Primary: common in queries, efficiency (one tuple/key)
  - Secondary: more useful when “almost a key”
- Clustered/Unclustered
  - Clustered:
    - fewer data page read, can have sparse index
    - expensive to maintain, at most one per file
- Dense/Sparse
  - Sparse: smaller, only for clustered index, at most one per file
  - Dense: multiple dense indexes, useful in some optimization (inverted data file)
- How to decide which indexes to create
  - Overhead (read/write index page, updates, deletions)
  - Depends on workload (Example in sec 8.4)

Multiple Levels of Index



- Useful when index file is big and is divided into multiple pages
- Efficient and standard implementation: B+ trees
  - balanced, good for both range and search query

- Tomorrow – Lec 6:
  - B+ Trees
  - Hash-based Index
    - Not good for range query