1) Conceptual Design (ER Diagram)

2) Logical Design (eg Relational Model)

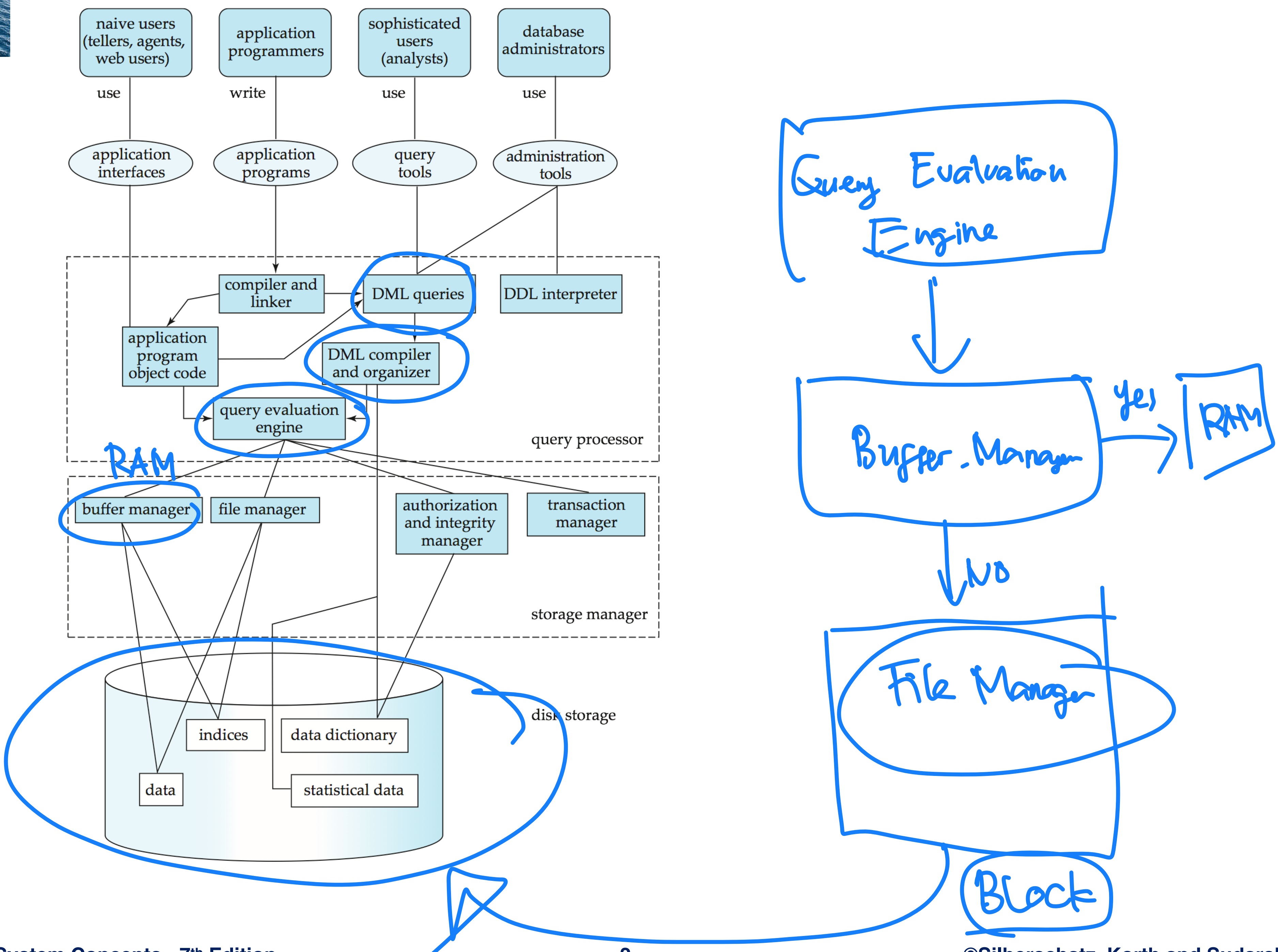
3) Physical Design

Topic 4 – Part 1 (File Organization)
Chapter 13, 14

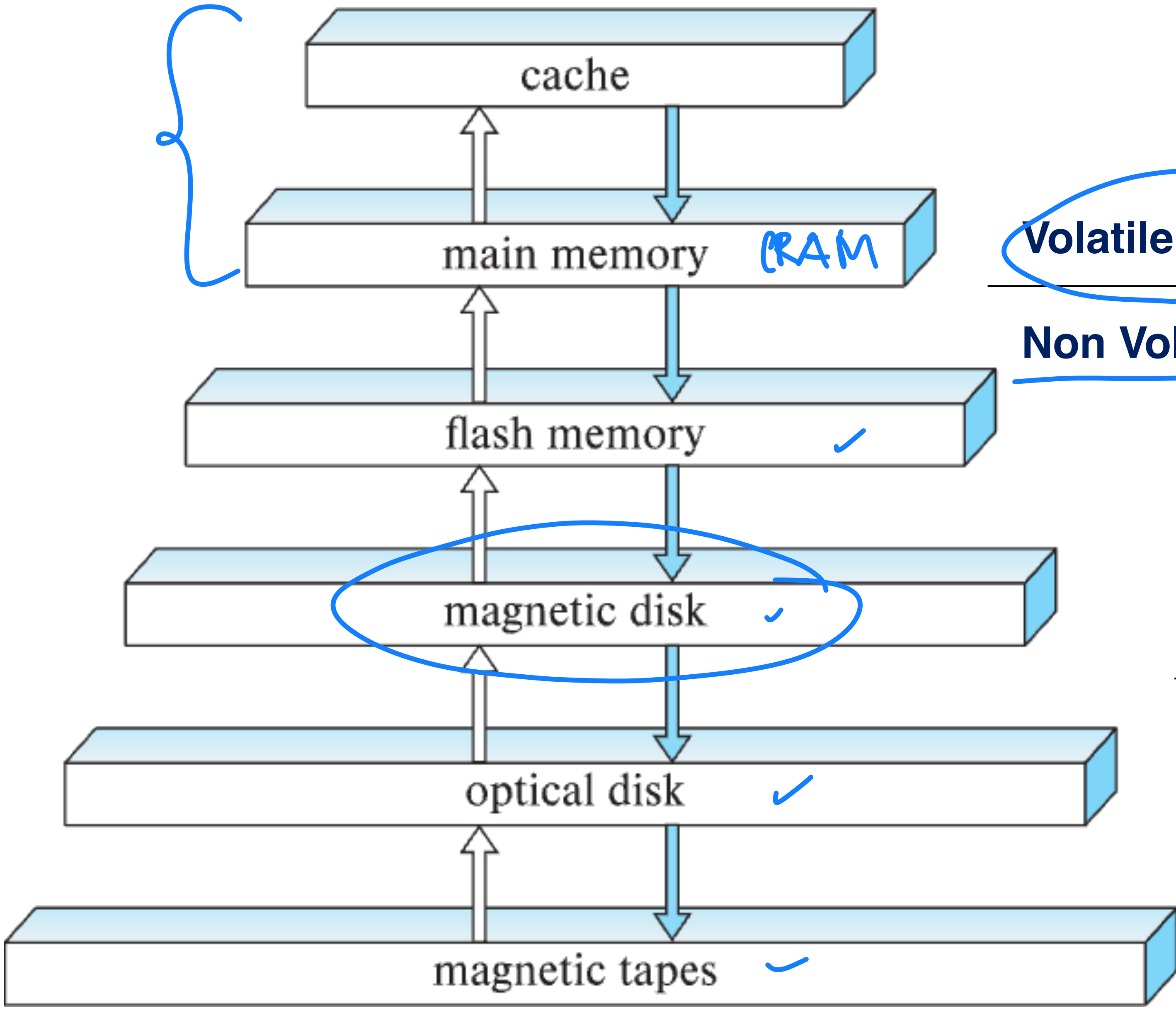
Indexing, Hashing

Database System Concepts, 7th Ed.

Database System Internals



Storage Hierarchy



Primary Storage

Volatile Storage

Non Volatile Storage

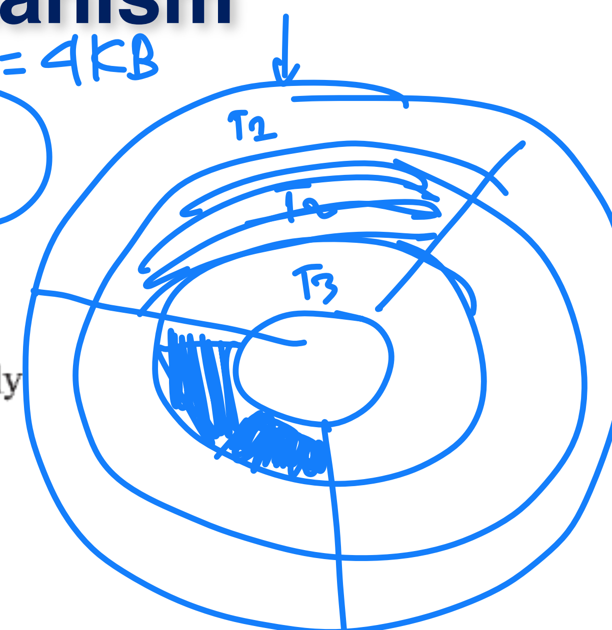
Secondary Storage

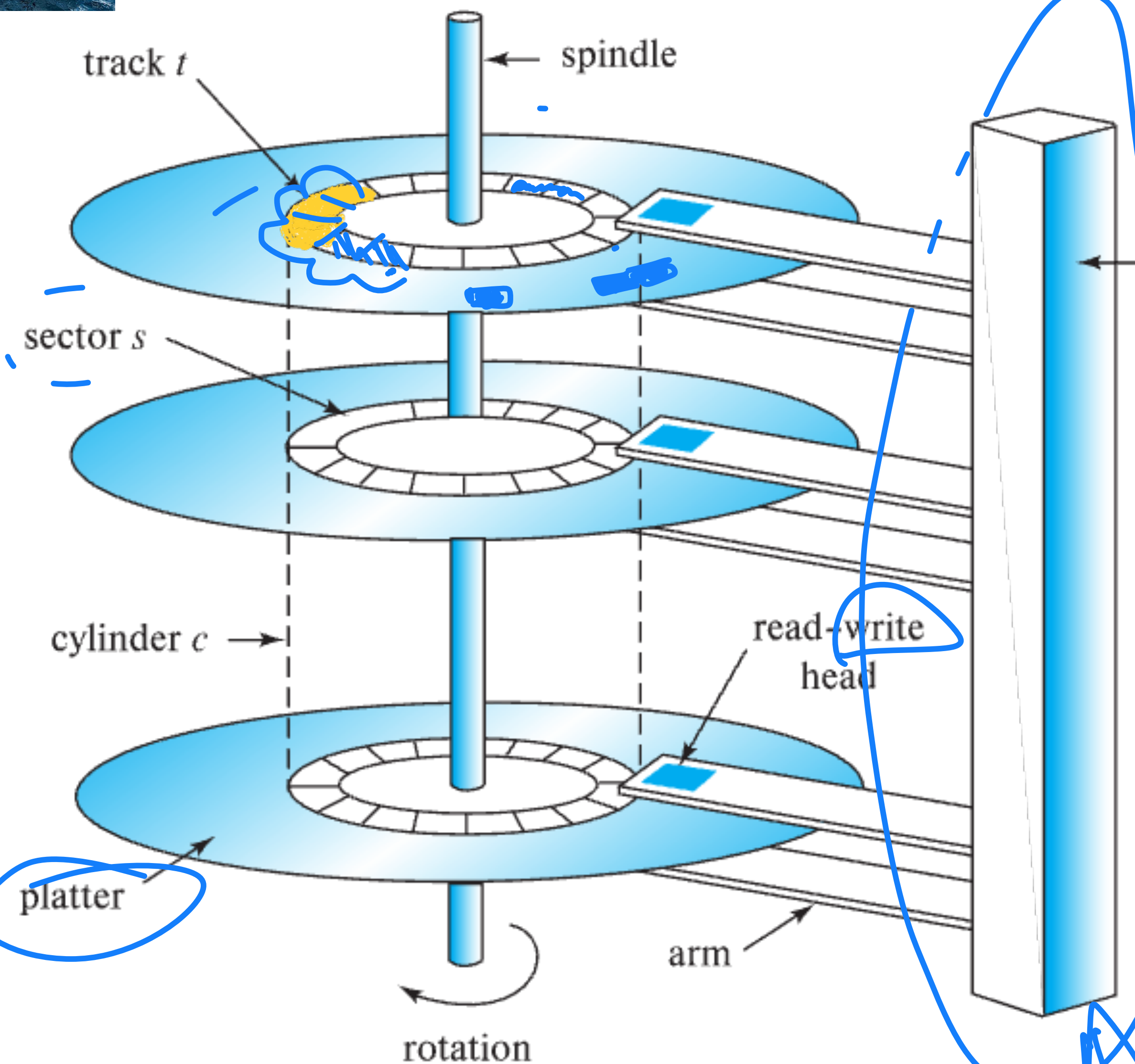
Tertiary Storage

Magnetic Hard Disk Mechanism

Sector = block = 4KB

20KB





Schematic diagram of magnetic disk drive

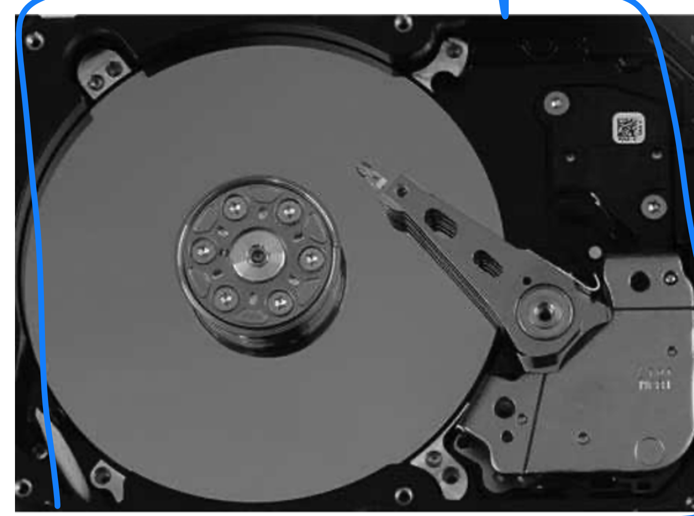


Photo of magnetic disk drive

Magnetic Hard Disk (Cont.)

20 KBytes

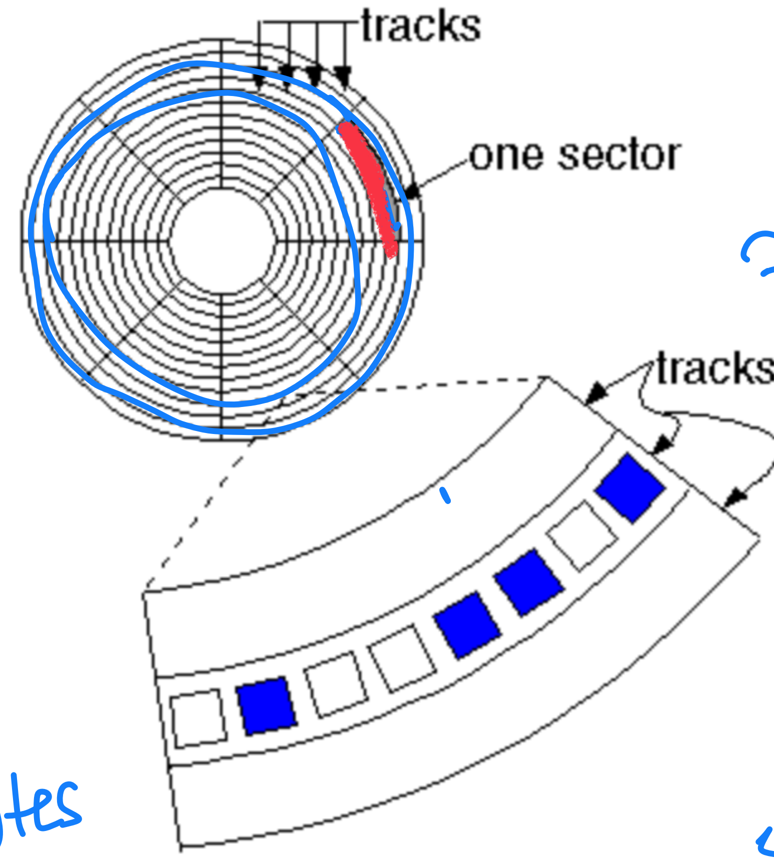
Sector as the smallest physical storage unit on a disk

512 bytes

4 Kbytes

⇒ 4096 bytes

1 KB = 1024 bytes



⇒ block → 4 KBytes

logical unit for storage allocation & retrieval

4 I/O

Tracks and Sectors

Tracks are concentric circles that are broken up into storage units called "sectors," typically 4,096 bytes long. The sector is the smallest unit that can be read or written. Tracks are only 75 nanometers wide today, and bit density is greater than one terabit per square inch. See areal density.

Source: https://www.pcmag.com/encyclopedia/term/magnetic-disk

Performance Measures of Disks

I/O operation

- **Access time** – the time it takes from when a read or write request is issued to when data transfer begins. Consists of:
  - **Seek time** – time it takes to reposition the arm over the correct track.
    - Average seek time is 1/2 the worst case seek time.
      - Would be 1/3 if all tracks had the same number of sectors, and we ignore the time to start and stop arm movement
    - 4 to 10 milliseconds on typical disks
  - **Rotational latency** – time it takes for the sector to be accessed to appear under the head.
    - 4 to 11 milliseconds on typical disks (5400 to 15000 r.p.m.)
    - Average latency is 1/2 of the above latency.
  - Overall latency is 5 to 20 msec depending on disk model
- **Data-transfer rate** – the rate at which data can be retrieved from or stored to the disk.
  - 25 to 200 MB per second max rate, lower for inner tracks

Performance Measures (Cont.)

4KB

- **Disk block** is a logical unit for storage allocation and retrieval (data transfer)
  - 4 to 16 kilobytes typically

    16KB 20 KB

    - Smaller blocks: more transfers from disk

      5 I/O

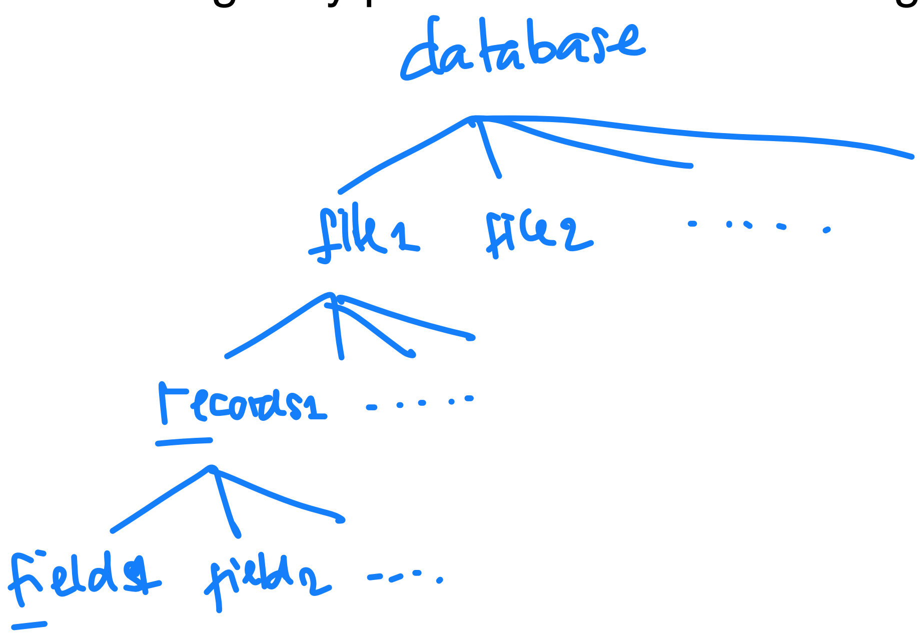
    - Larger blocks: more space wasted due to partially filled blocks

      2 I/O

- **Sequential access pattern**
  - Successive requests are for successive disk blocks
  - Disk seek required only for first block
- **Random access pattern**
  - Successive requests are for blocks that can be anywhere on disk
  - Each access requires a seek
  - Transfer rates are low since a lot of time is wasted in seeks
- **I/O operations per second (IOPS)**
  - Number of random block reads that a disk can support per second
  - 50 to 200 IOPS on current generation magnetic disks

File Organization

- The database is stored as a collection of *files*.
- Each file is a sequence of *records.*
- A record is a sequence of fields.
- Each file is also logically partitioned into fixed-length storage units called block.



File Organization (Cont.)

- One approach of organizing data:
  - Assume record size is fixed
  - Each file has records of one particular type only
  - Different files are used for different relations

  This case is easiest to implement; will consider variable length records later
- We assume that records are smaller than a disk block

4KB

a record size << 4KB

file → table / relation

record → one tuple / row

field → attribute / column value

Fixed-Length Records

Block size : 4KB = 4096 Bytes

- Simple approach:
  - Store record *i* starting from byte $n * (i - 1)$, where *n* is the size of each record.

    100 * (40 - 1) = 3900
  - Record access is simple but records may cross blocks
    - Modification: do not allow records to cross block boundaries

instructor (ID, name, dept_name, salary)

100 bytes

4096 // 100 = 40 records

96 bytes

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

i:1 = 0

i:2 = 100

i:3 = 200

i:40 3900

i = 41

96

4 byte another block

Fixed-Length Records

- Deletion of record *i:* alternatives*:*
  - **move records *i* + 1, . . ., *n* to *i*, . . . , *n* – 1**
  - move record *n* to *i*
  - do not move records, but link all free records on a *free list*

Record 3 deleted

i i''

~~record 0~~	~~10101~~	~~Srinivasan~~	~~Comp. Sci.~~	~~65000~~
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

$O(n)$

Fixed-Length Records

- Deletion of record *i:* alternatives*:*
  - move records $i + 1, \ldots, n$ to $i, \ldots, n - 1$
  - **move record *n* to *i***
  - do not move records, but link all free records on a *free list*

Record 3 deleted and replaced by record 11

record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 11	98345	Kim	Elec. Eng.	80000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000

Fixed-Length Records

- Deletion of record *i*: alternatives:
  - move records $i + 1, \ldots, n$ to $i, \ldots, n - 1$
  - move record n to i
  - **do not move records, but link all free records on a *free list***

Block 1

header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000

25

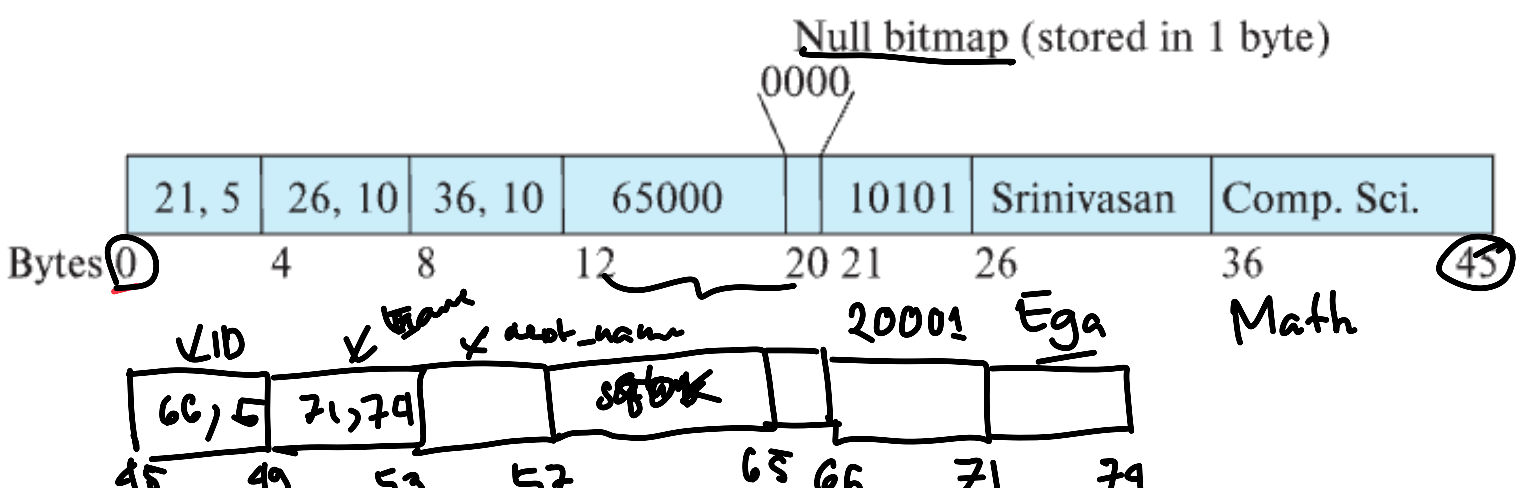
100 bytes

Block 2

1200 Bytes

Variable-Length Records

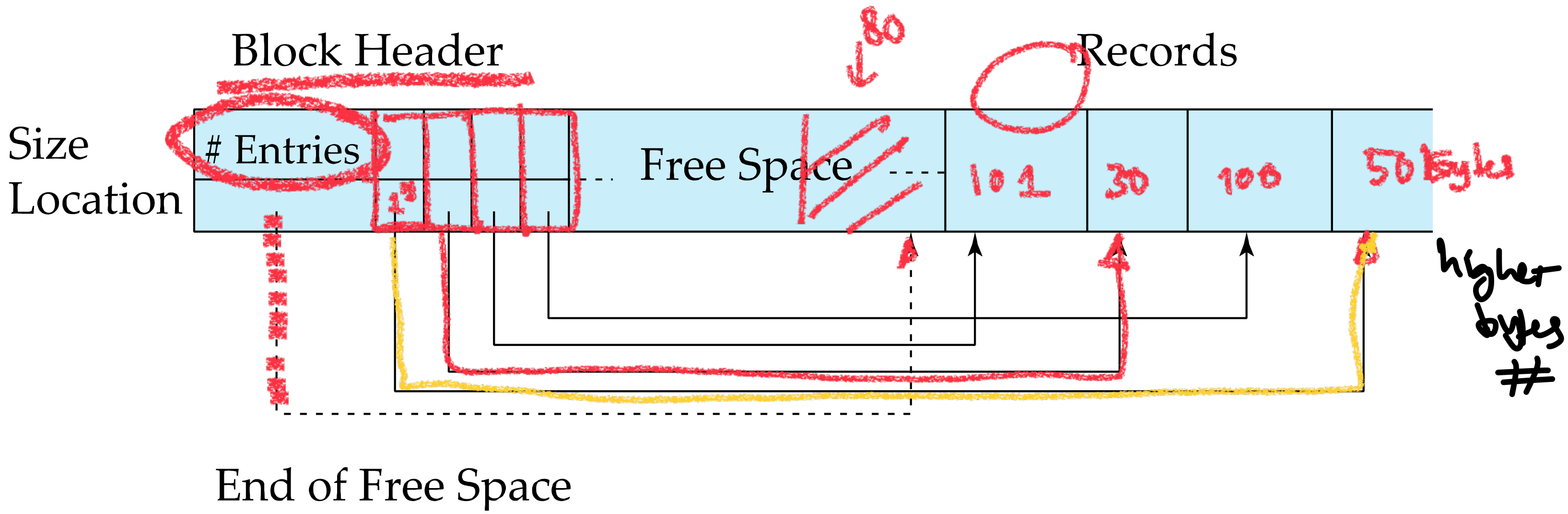
- Variable-length records arise in database systems in several ways:
  - Storage of multiple record types in a file.
  - Record types that allow variable lengths for one or more fields such as strings (**varchar**)
  - Record types that allow repeating fields (used in some older data models).
- Attributes are stored in order
- Variable length attributes represented by fixed size **(offset, length)**, with actual data stored after all fixed length attributes
- Null values represented by null-value bitmap



```
create table instructor (
    ID varchar(5),
    name varchar(50),
    dept_name varchar(25),
    salary int,
    primary key (10),
    foreign key (dept_name) references (department)
)
```

1 char → 1 byte

Variable-Length Records: Slotted Page Structure



- **Slotted page** header contains:
  - number of record entries
  - end of free space in the block
  - an array containing location and size of each record
- Records can be moved around within a page to keep them contiguous with no empty space between them; entry in the header must be updated.
- Pointers should not point directly to record — instead they should point to the entry for the record in header.

Storing Large Objects

4KB

binary large object

- Databases often store data that can be much larger than a disk block.
- E.g., blob/clob types

images /video

character large object

5MB

- Records must be smaller than pages
- Alternatives:
  - Store as files in file systems → OS
  - Store as files managed by database
  - Break into pieces and store in multiple tuples in separate relation
    - PostgreSQL TOAST

1 file ~> 1 relation

Topic 4 Poll 1

4096/100 = 40 records
96 bytes

- Suppose a disk has a block size of 4 KB (4096 bytes). A file contains a database relation with 50 fixed-size records, each 100 bytes. How many I/O operations are required to retrieve all data from the relation?

instructor
50 tuples

select *
from instructor

50x100 = 5000 bytes

Block 0
record 1
⋮
record 40

2 I/O operation

Block 1
record 41
⋮
50

Organization of Records in Files

- **Heap** – record can be placed anywhere in the file where there is space
- **Sequential** – store records in sequential order, based on the value of the search key of each record
- In a **multitable clustering file organization** records of several different relations can be stored in the same file
  - Motivation: store related records on the same block to minimize I/O
- **B+-tree file organization**
  - Ordered storage even with inserts/deletes
  - More on this in Chapter 14 (Topic 4 Part 2)
- **Hashing** – a hash function computed on search key; the result specifies in which block of the file the record should be placed
  - More on this in Chapter 14 (Topic 4 Part 2)

Heap File Organization

- Records can be placed anywhere in the file where there is free space
- Records usually do not move once allocated
- Important to be able to efficiently find free space within file
- **Free-space map** → **data structure that tracks the fraction of the free space in each block**
  - Array with 1 entry per block. Each entry is a few bits to a byte, and records fraction of block that is free

    2^3
  - In example below, 3 bits per block, value divided by 8 indicates fraction of block that is free

    Block is $\frac{6}{8}$

    1st leve

    0 ... 16

    6 blocks

  - Can have s

4	2	1	4	7	3	6	5	1	2	0	1	1	0	5	6

  - In example below, each entry stores maximum from 4 entries of first-level free-space map

    2nd level

4	7	2	6

- Free space map written to disk periodically, OK to have wrong (old) values for some entries (will be detected and fixed)

Sequential File Organization

- Suitable for applications that require sequential processing of the entire file → relation / table
- The records in the file are ordered by a search-key

ID

0	10101	Srinivasan	Comp. Sci.	65000	
1	12121	Wu	Finance	90000	
	15151	Mozart	Music	40000	
	22222	Einstein	Physics	95000	
	32343	El Said	History	60000	
	33456	Gold	Physics	87000	
	45565	Katz	Comp. Sci.	75000	
	58583	Califieri	History	62000	
	76543	Singh	Finance	80000	
	76766	Crick	Biology	72000	
	83821	Brandt	Comp. Sci.	92000	
	98345	Kim	Elec. Eng.	80000	

instructor

Sequential File Organization (Cont.)

- Deletion – use pointer chains
- Insertion –locate the position where the record is to be inserted
  - if there is free space insert there
  - if no free space, insert the record in an overflow block
  - In either case, pointer chain must be updated
- Need to reorganize the file from time to time to restore sequential order

Block 0

ID				
10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

Block 1

32222	Verdi	Music	48000	

Multitable Clustering File Organization

Store several relations in one file using a **multitable clustering** file organization

department

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Physics	Watson	70000

instructor

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

Select *
from department,
instructor
where department.
dept_name
=
instructor.
dept_name

multitable clustering of *department* and *instructor*

Comp. Sci.	Taylor	100000	
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000
Physics	Watson	70000	
33456	Gold	Physics	87000

→ dept

Multitable Clustering File Organization (cont.)

join

- good for queries involving *department* ⋈ *instructor*, and for queries involving one single department and its instructors
- bad for queries involving only *department*
- results in variable size records
- Can add pointer chains to link records of a particular relation

Partitioning

- **Table partitioning**: Records in a relation can be partitioned into smaller relations that are stored separately
- E.g., *transaction* relation may be partitioned into *transaction_2018, transaction_2019, etc.*

credit_card_usage

- Queries written on *transaction* must access records in all partitions
  - Unless query has a selection such as *year*=2019, in which case only one partition in needed
- Partitioning
  - Reduces costs of some operations such as free space management
  - Allows different partitions to be stored on different storage devices
    - E.g., *transaction* partition for current year on SSD, for older years on magnetic disk

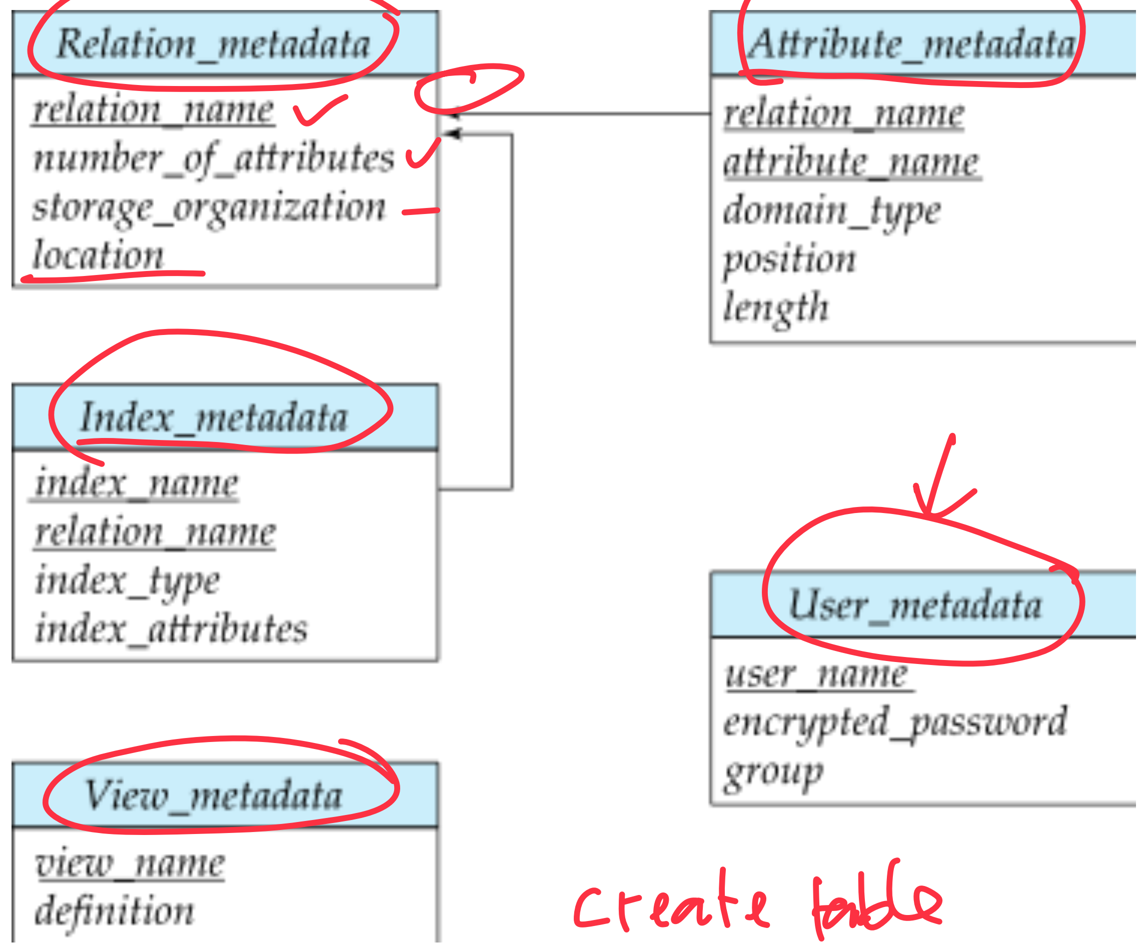
Data Dictionary Storage

The **Data dictionary** (also called **system catalog**) stores **metadata**; that is, data about data, such as

- Information about relations
  - names of relations
  - names, types and lengths of attributes of each relation
  - names and definitions of views
  - integrity constraints
- User and accounting information, including passwords
- Statistical and descriptive data
  - number of tuples in each relation
- Physical file organization information
  - How relation is stored (sequential/hash/…)
  - Physical location of relation
- Information about indices (Chapter 14)

Relational Representation of System Metadata

- Relational representation on disk
- Specialized data structures designed for efficient access, in memory



create table

view

End of Topic 4 Part 1